THE UNIVERSITY OF ALBERTA

FAULT DETECTION IN SEQUENTIAL CIRCUITS

by

© VLADIMIR FRANTISEK BERKA

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE
STUDIES AND RESEARCH IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE
OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE
EDMONTON, ALBERTA
SPRING, 1975

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled "Fault Detection in Sequential Circuits", submitted by Vladimir Frantisek Berka, in partial fulfillment of the requirements for the degree of Master of Science.

Wayne A. Davis
Supervisor

Keith A. Stromsmoe

Date 6 May 1975

ABSTRACT

This thesis is concerned with the problem of testing of sequential machines. Initially testing methods which represent the current state of art are surveyed. Then a testing method which can be used to generate testing sequences for an unrestricted class of machines is given. Different types of logical faults are defined and methods which find tests for individual faults or a combination of faults are provided. The resulting testing sequences, while not minimal, compare favourably with the sequences obtained by current methods. Also a procedure which can be used to translate the effects of faults on a state table is described. This procedure makes the sequence generation process more convenient and reduces the amount of information which has to be kept about the behaviour of individual faults. Finally, fault diagnosis is discussed and method which can be used to generate a diagnostic testing sequence is given.

ACKNOWLEDGEMENT

The author wishes to express sincere appreciation to his supervisor, Dr. Wayne A. Davis, for guidance throughout the period of research and writing of this thesis. The author also wishes to thank Computing Services and The Department of Chemical Engineering of The University of Alberta for allowing him to work on this paper while being employed.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

1.1 Assumptions and Definitions 2

1.2 Notation 8

II. A SURVEY OF PREVIOUS RESULTS 10

2.1 Circuit testing approach 11

2.1.1 Poage-McCluskey's method 15

2.1.2 Path sensitizing method 28

2.2 Machine identification approach 32

2.2.1 Hennie's method 33

2.2.2 Hennies-Hsieh's method 42

2.2.3 Gonenc's method 53

2.3 Hybrid approach 60

2.3.1 Kohavi's method 61

III. TESTING SEQUENCES 66

3.1 Fault Classification 66

3.2 Testing for output faults 76

3.3 Testing sequences for machines with non-overlapping state faults 82

3.4 Testing sequences for machines with overlapping state faults 98

Chapter Page

IV. GENERATION OF TESTING SEQUENCES 104

4.1 Fault set identification 105

4.2 Generation of state propagating sequences 116

V. FAULT DIAGNOSIS 120

VI. CONCLUSIONS 128

REFRENCES 131

APPENDIX 134

LIST OF TABLES

Table Page

2.1 Sequential table for machine M2.2 22

2.2 Rearranged sequential table machine M2.2 24

2.3 Modified sequential table for machine M2.2 . 26

2.4 Maps for mahine M2.3 62

3.1 State tables of machines m(0)...m(6) 75

3.2 State table of machine M3.2 81

3.3 State tables of machines m(0),...,m(6) 85

4.1 State table of machine M4.1 110

4.2 Test table of machine M4.1 111

4.3 State and output fault dictionary for M4.1 116

4.4 State and output fault dictionary for M4.1 116

LIST OF FIGURES

Figure Page

1.1 Sequential machine 6

2.1 Circuit diagram of machine M2.1 13

2.2 Circuit diagram of machine M2.2 18

2.3 Successor tree for machine M2.2 23

2.4 Sequential machine decomposed into a set of combinational circuits 30

2.5 Homing tree for machines m(0) and m(1) 35

2.6 Predecessor tree for machine m(0) 39

2.7 T-diagram for Example 2.6 59

3.1 Sequential machine 68

3.2 Circuit diagram of machine M3.1 74

3.3 Circuit diagram and state table of M3.2 ... 80

3.4 A testing tree 88

3.5 Testing tree for machine M3.2 90

3.6 Circuit diagram for machine M3.3 95

3.7 Testing tree for M3.3 97

3.8 A testing tree 99

3.9 Testing tree for machine M3.3 102

4.1 Circuit diagram of machine M4.1 109

4.2 Testing tree of machine M4.1 113

Figure Page

4.3 Testing tree for machine M4.1 114

5.1 Testing tree for machine M4.1 124

5.2 Testing tree for machine M4.1 126

Chapter 1

INTRODUCTION

This thesis deals with sequential circuit testing, particularly with the generation of tests which detect and locate faults in such circuits. Since every digital system is essentially a large sequential circuit or a set of sequential circuits, testing is important in order to improve reliability and to ensure correct operation. Even in systems where the reliability is improved by the use of redundant fault-masking components, fault detection procedures are definitely useful.

These systems can be conceptually divided into primary and secondary parts, where the secondary part is assumed to be the fault masking part. In such systems, a malfunction will only occur if a fault in the primary system is followed by another fault, this time in the redundant fault-masking part, before the fault in the primary system is discovered and repaired. Naturally, the same result would be obtained if the fault in the fault-masking component is followed by another fault in the primary system. To ensure correct operation of the system over a long period of time, it is important that faults in the primary system are detected and repaired in a period of time short enough to make the simultaneous occurence of faults in both primary and secondary systems highly unlikely.

In principle, to make the testing of fault-tolerant systems possible, they should be designed in such a way that it is possible to disable the fault-masking for a short period of time in which the testing of the primary system can take place. This can be implemented by allowing tests to access outputs independent of the fault masking. It should also be possible to test the secondary system in a similar manner. Upon completion of the fault-detecting program, the fault-masking should again be enabled. If the result of the test indicates the presence of a fault then the fault-locating procedures should start immediately. The execution of fault-locating programs will again require that the fault-masking is disabled on systems with this feature. In other words, while a fault-detecting experiment determines if a fault is present in a system, the fault-locating experiment will determine which component is faulty.

1.1 Assumptions and Definitions.

Faults which occur in the electrical components of a sequential circuit, called physical faults, can influence the function of the circuit in number of different ways. Those which influence the logical behaviour of the circuit are called logical faults. In this thesis, only logical faults are considered. A fault may be present permanently or it may be a fault of a temporary nature. A permanent fault influences the logical function of the circuit in a fixed way, until it is repaired. A temporary or intermittent fault, sometimes changes the logical behaviour of the circuit yet sometimes allows the circuit to operate correctly. This thesis is concerned with permanent faults, although a possible extension to include intermittent faults is briefly discussed in section 3.4.

Permanent logical faults can be caused by a number of different physical malfunctions. Two types of physical faults, however, are predominant:

1) Stuck type fault.
2) Bridging fault.

Stuck type faults maintain constant voltages at affected connections in the circuit. These voltages correspond to either a logical one or a logical zero. In the first case the fault is called a "stuck-at-one" or s-1 fault, and in the second case it is called a "stuck-at-zero" or s-0 fault.

Bridging faults are caused by shorts between two or more input leads or between the input and output of the circuit. More generally, one can say that the first type of bridging fault occurs when two independent paths of the circuit are interconnected, while the second occurs when two different points on the same path are accidentally connected. The first type is called an *input bridging fault* [10] while the second is called a *feedback bridging fault* [10].

An input sequence designed for the purpose of testing a *synchronous* sequential circuit, will be called a *testing sequence*. A testing sequence consists of a number of input vectors, where each input vector is a set of values applied, at the same time, to the inputs of the circuit. Testing sequences which are designed to detect the presence of faults in the circuit will be called *fault-detecting sequences* while sequences which are designed to identify faulty components of the circuit will be called *fault-locating or diagnosing sequences*.

In sequential machines, there are two known approaches to fault detection:

1) The *circuit testing* approach.
2) The *machine identification* approach.

In some test generation methods, these approaches are combined into a *hybrid* approach. Here the circuit testing approach is used to determine which transitions of a machine are influenced by a given set of faults. Then the testing sequence is generated using the machine identification approach, but the subsequences which test for the correctness of transitions which cannot be influenced by a given set of faults are not included in the final testing sequence. In Chapter 2, fault detection methods are surveyed and classified by the above approaches. In Chapters 3 and 4, a new hybrid method is proposed. In this method, testing sequences are developed by forcing a machine into a fault-sensitive situation and examining every possible outcome under any of the possible faults. This procedure compares favourably with other methods, with respect to the length of a testing sequence and the effort needed to obtain one.

Every sequential machine, in principle, consists of a combinational circuit, which has some of its outputs connected via a feedback loop to some of its inputs, as pictured in Fig. 1.1. Some of the methods for the generation of tests for sequential circuits, require tests to be obtained for faults in the combinational part of the sequential machine. A fault test is an input, which, when applied to the combinational circuit, will result in one output if the fault does not occur, and in a different output if the fault is present. A complete set of tests for an arbitrary set of faults contains at least one test for each fault contained in the set. Numerous methods for finding complete sets of tests have been developed [1,2,4,5,8, 9,15,17,19,20,23,27].

It is assumed, throughout, that the known methods with which one can generate a complete set of tests for the combinational part of the sequential machine will be exploited.

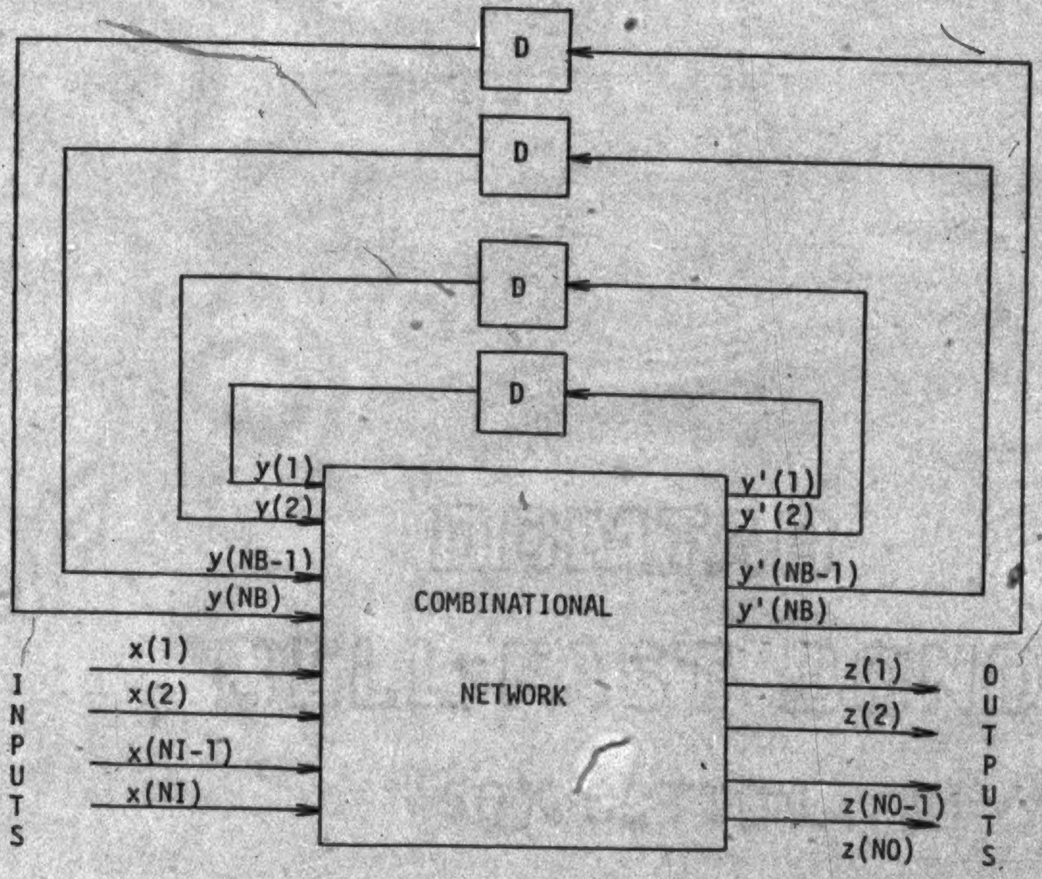

D
D
D
D
y(1)
y(2)
y(NB-1)
y(NB)
y'(1)
y'(2)
y'(NB-1)
y'(NB)
COMBINATIONAL
NETWORK
INPUTS
x(1)
x(2)
x(NI-1)
x(NI)
z(1)
z(2)
z(NO-1)
z(NO)
OUTPUTS

Fig. 1.1 Sequential circuit

It is also assumed that the reader is familiar with the basic principles involved in the testing of combinational circuits. Briefly these are:

1) Path sensitizing, described in [1,3,8,9], is a scheme under which, a circuit is tested by sensitizing essential paths between inputs and outputs of the circuit. To sensitize a path for a gate fault, it is first necessary to find a test for that fault. Then the inputs of the gates which have their outputs connected to this gate are calculated so that their outputs yield the required test. This is repeated until the inputs to the circuit are reached and the necessary input test therefore obtained. It is also necessary to sensitize the path between the gate's output and at least one output of the circuit in order to propagate the test result to the circuit's output.

2) The equivalent normal form (ENF), also called equivalent sum of products (ESP), defined and discussed in [1,3,8,9,19], is a scheme under which any irredundant combinational circuit can be converted into a two-level combinational circuit, where the first level contains only "AND" gates and the second level consists of a single "OR" gate. The logical function of the resulting circuit can be expressed as a logical sum of logical products, hence the name. It is a property of the ESP that a complete set of tests for its input lines is also a complete set of tests for all single stuck type faults in the original circuit [1]. In addition, a complete set of tests for single

faults also detects all multiple faults, provided that the network is irredundant [8]. A more detailed discussion of multiple faults can be found in [5,8,20].

1.2 Notation

The following notation will be used in this thesis:

1) Elements, such as faults, inputs, etc, will be denoted with lower case letters, with or without subscripts. For example, f will represent any single fault. To emphasize that a fault is a member of some larger set it will be written as f(i), meaning the i-th fault of the set F. Other commonly used symbols are:

s(i) - the i-th internal state.
t(i,j,k) - the k-th input of the j-th test for fault f(i).

2) Sets, vectors and sequences will be denoted by upper case letters, again with or without subscripts. Therefore F and F(i) will stand for a set or subset of faults. Symbols used are:

F - the set of faults.
S(i) - the state assignment of the internal state s(i).
U(i) - the input vector.
T - the complete set of tests.
T(i) - a set of tests for f(i).

T(i,j) - the j-th test for f(i).

A comma "," will be used to indicate concatenation.

3) Functions will be denoted with two letter sequences of capitals. The following fuctions will be used:

NF - the number of faults f in a set of faults.

NF(i) - the number of faults in the i-th subset.

NT(F) - number of tests available to test faults in F.

NT(i) - the number of tests for f(i).

LT(F) - the length of the testing sequence.

NS - the number of states in a sequential circuit.

NI - the number of inputs in a sequential circuit.

NB - the number of feedback lines in a sequential circuit.

Chapter 2

A SURVEY OF PREVIOUS RESULTS

In this chapter a survey which describes methods for testing of sequential circuits, published after 1964, is presented. The problem of testing sequential machines is, in principle, a problem of distinguishing, by means of either a preset or an adaptive experiment, between the machine which functions properly and the one which does not. In the experiment, a testing sequence is applied to the machine and, depending on the response, the machine is found to be either operating correctly or not. Naturally, the testing sequence should be as short as possible and it should also be easy to obtain.

There are three known approaches to the problem of of the generation of testing sequences in sequential circuits. These approaches differ in the assumptions which are made about a machine, in the effort required to generate a testing sequence, and in the length of the testing sequence obtained.

The first methods developed, belong to a circuit testing approach [3,9]. Under this approach the function of a machine without any fault present is compared to the function of the same machine but with individual faults present, one at a time. Then the inputs which result in different outputs in the good and faulty machines are selected in the best possible way. In the machine identification approach [13,14,9] the function of a good machine is compared to the function of all possible machines which have the same or smaller number of states. This is done regardless of whether or not some of the faulty machines can actually result from any circuit fault. Finally the hybrid approach is a combination of the other two. In this approach the testing sequence is initially obtained according to the machine identification scheme, then some inputs are eliminated, because the faults which they test for, cannot occur under the assumptions made for a fault set. In the following sections, the above three approaches and specific methods developed according to them will be described in detail.

2.1 Circuit testing approach

In this approach the first step is to obtain a state table from the given circuit. Then assuming the presence of a fault belonging to a specific set, a state table describing the function of the faulty machine is obtained. This process is repeated for every fault in the set. Then the input sequence which results in an output which is different for good and faulty machines is found. These sequences can be obtained for every possible pair of good and faulty machines and then merged into a testing sequence. Alternatively, all possible sequences can be applied to all machines and the one which will lead to different outputs for the good machine and other faulty machines in the shortest number of steps is selected. The following example will illustrate an application of the circuit testing approach.

Example 2.1: Find a testing sequence for the machine given in Fig. 2.1 if the only possible fault is s-0 at the location 1.

Assuming that the state y=0 will be called A, and the state y=1 will be called B, the state tables T(0) and T(1) describing the "good" machine m(0) and faulty machine m(1) are given in the following tables:

T(0)

y/x	0	1
A	A,10	B,01
B	B,01	A,10

T(1)

y/x	0	1
A	A,10	A,10
B	B,01	A,10

The input 1 can leave machine m(0) in either A or B depending on initial state of m(0) but machine m(1) will be certainly in state A. Assume for a while that machine m(0) was initially in

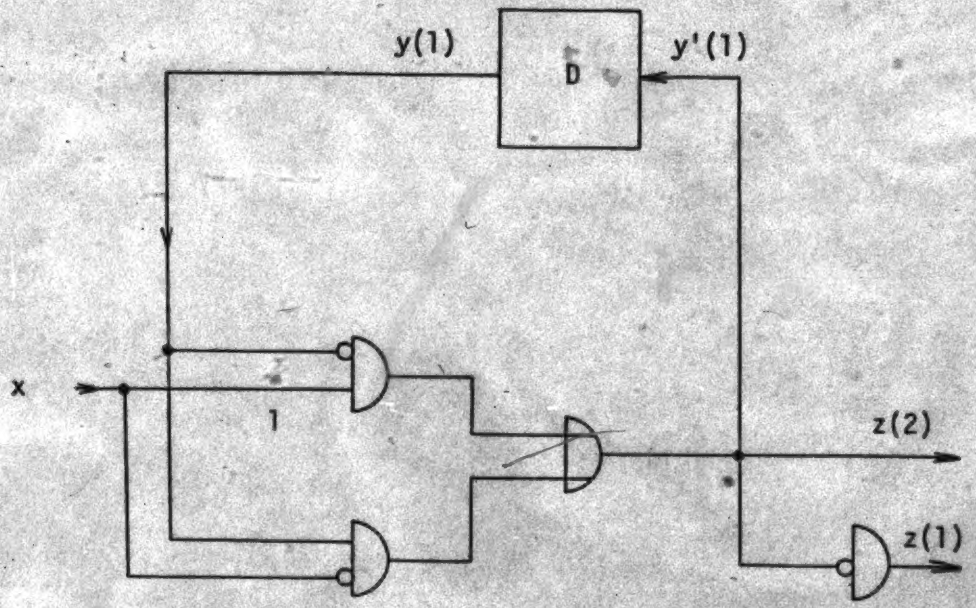

y(1)
D
y'(1)
x
1
z(2)
z(1)

Fig. 2.1 Circuit diagram of machine M2.1

state A. Then the sequence 1 will distinguish between m(0) and m(1) as is apparent from the following:

	Input		1	
Machine 0	State	A		B
	Output		01	
Machine 1	State	-		A
	Output		10	

The machine m(0) however could have been also in the state B when the sequence was applied. In that case the following would happen:

	Input		1	
Machine 0	State	B		A
	Output		10	
Machine 1	State	-		A
	Output		10	

The machines would not be distinguished by the sequence developed under the initial assumption. However in both cases machine m(0) will certainly be in state A after the application of the input sequence 1 and another input 1 will distinguish between T(0) and T(1) in both cases:

	Input		1		1	
Machine 0	State	B		A		B
	Output		10		01	
Machine 1	State	-		A		A
	Output		10		10	

As a result the sequence 1,1 is a testing sequence for machine m(0), which will detect the fault f(1) regardless of the initial state.

In the following sections specific methods which represent the circuit testing approach will be described.

2.1.1 POAGE-McCLUSKEYs method [3,9]

In this method the state tables of good and faulty machines are systematically developed and then are manipulated in such a way that the shortest possible testing sequence can be selected, relatively conveniently from all possibilities.

The following assumptions are made:

1) The circuit diagram is known.

2) Both "good" and faulty machines can be reset to a known initial state.

3) Only stuck type faults are considered.

The detailed algorithm is then given by the following:

1) Find the state tables for the given machine m(0) and for all faulty machines m(1),...,m(NF). Since the machines' circuitry is known this is a straightforward task. It is suggested that the circuit equations be written in such a way that they reflect the operation of the machine under all possible faults. To do so, new variables called fault parameters are incorporated into the equations, producing expressions called literal propositions. The setting of fault parameters changes the equations to represent the operation of

the circuit under a given fault, see Example 2.2 or [23,3].

2) Construct the product tables P(0,1),...,P(0,NF). The product table P(0,i), for the pair m(0),m(i), represents the operation of machines m(0) and m(i) in parallel. If the outputs of their corresponding transitions differ, an "*" is entered into the table to mark the transition which can be used to differentiate between m(0) and m(i).

3) Form a table, which contains the successors of all states, entered by machines m(0),...,m(NF), under all possible inputs. The set of successors entered by all machines under a single input will be called a composite state, while the table will be called a composite sequential table. First the successors of the initial composite states are entered, then their successors, etc. The product tables obtained in step 2 are utilized in this process. In particular, if the transitions of m(0) and m(i) differ, which is indicated by an * in the P(0,i) product table, the asterisk is transfered to the composite table. The successor of an * state is always an * state. An * state can be considered equal to any other state and treated as a "don't care" state. This is because once the * state is reached for a certain machine the corresponding fault will be detected and there is no need to find other states in which the fault would be also detected. The development of the table continues until a composite state having *s for all

machines m(1),...,m(NF) is reached or until there are no new composite states produced. In the first case all faults can be detected by an experiment, while in the second case only the faults corresponding to machines with an * are detectable.

4) Construct the successor tree to find the shortest input sequence, from the initial composite state to the final composite state.

The following example will illustrate the above method.

Example 2.2: For the machine given by the circuit diagram in Fig. 2.2, find a testing sequence for the following set of faults.

Fault #	Fault location	Fault type
1	1	s-0
2	3	s-1
3	13	s-1
4	18	s-1
	20	s-0

The equations for the given circuit are:

$$y'(1)=y(2)+y(1)\overline{x(1)}+x(1)x(2)\overline{y(1)},$$

$$y'(2)=\overline{x(1)}\,\overline{x(2)}+x(1)\overline{y(1)},$$

$$z(1)=x(1)\overline{y(1)}+y(1)\overline{x(1)}+x(1)x(2).$$

The literal propositions can now be obtained; to illustrate consider function y(1):

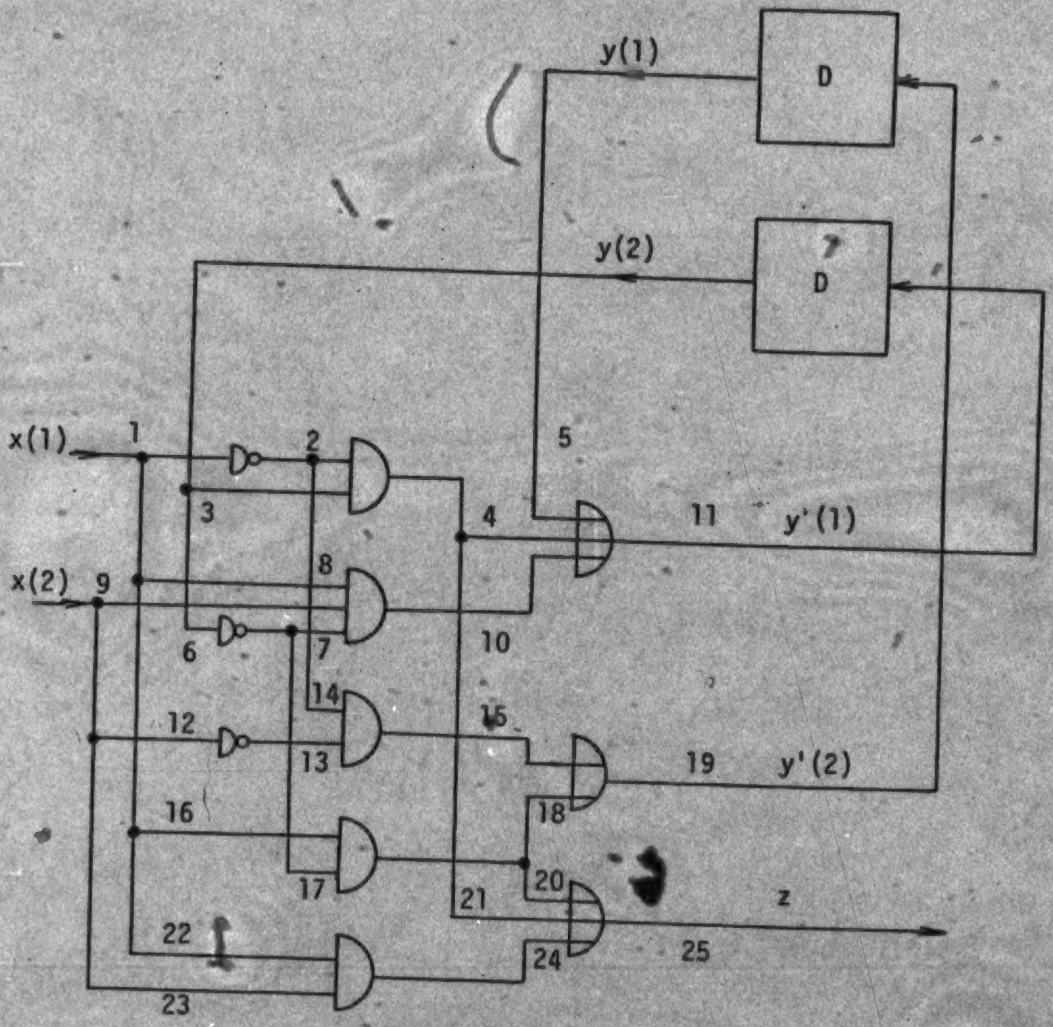

y(1)
D
y(2)
D
x(1)
1
2
3
4
5
11
y'(1)
x(2)
9
8
6
7
10
12
13
14
15
19
y'(2)
16
17
18
20
21
z
22
24
25
23

Fig. 2.2. Circuit diagram of machine M2.2.

Wire 1 will be a 1 if either x(1) is a 1 or if wire 1 is stuck at 1. The literal proposition for wire 1 is therefore

$$1 = x(1)[1(1)],$$

where 1(1) is a fault parameter which will be set to 1 if the particular fault is present or to 0 if complementary fault i.e. s-0 is present at location 1. If the fault parameter is assigned either 0 or 1 this value then replaces the input literal x(1) in the equation and the literal proposition reduces to the equation expressing the operation of the circuit under the given fault. The wire 2 will be in 1 if either wire 1 is in 0 or if wire 2 is stuck at 1. This can be written as:

$$2 = \overline{1}[2(1)].$$

The negated literal proposition can be obtained by complementing the input literal and changing the subscript of the fault parameter, i.e:

$$2 = \overline{x(1)}[1(0), 2(1)],$$

which can be interpreted that wire 2 will be a 1 if either x is 0 or wire 1 is stuck at 0 or if wire 2 is stuck at 1.
The following equations are obtained in a similar manner.

$$3 = y(1)[3(1)],$$

$$4 = 2[4(1)]\ 3[4(1)] = x(1)[1(0), 2(1), 4(1)]\ y(1)[3(1), 4(1)],$$

$$5 = y(2)[5(1)],$$

$6 = y(1)[6(1)],$

$7 = \overline{y(1)}[6(0), 7(1)],$

$8 = x(1)[8(1)],$

$9 = x(2)[9(1)],$

$10 = y(1)[6(0), 7(1)] . x(1)[8(1)] . x(2)[9(1)],$

$$\begin{aligned} 11 = {} & y(2)[5(1), 11(1)] + \overline{x(1)}[1(0), 2(1), 4(1), 11(1)] \\ & y(1)[3(1), 4(1), 11(1)] + \\ & \overline{y(1)}[6(0), 7(1), 8(1), 11(1)] \\ & x(1)[8(1), 10(1), 11(1)] + x(2)[9(1), 10(1), 11(1)], \end{aligned}$$

$y'(1) = 11.$

The literal propositions can be determined directly from the circuit diagram by following the path leading from the input to the particular output z or internal variable y and noting all wires which can influence it. Then a 0 or 1 is appended to each number of each wire, depending on the type of fault. The remaining literal propositions are:

$$\begin{aligned} y'(2) = {} & \overline{x(2)}[9(0), 13(1), 15(1), 19(1)] \ \overline{x(1)}[1(0), 14(1), 15(1), 19(1)] + \\ & x(1)[16(1), 18(1), 19(1)] \ \overline{y(1)}[6(0), 17(1), 18(1), 19(1)] \end{aligned}$$

BELL-FAST BOND
50% COTTON

$z(1)=\overline{y(1)}[3(0),7(1),17(1),20(1)]$

$x(1)[1(1),16(1),20(1)]+y(1)[3(1),4(1),21(1)]$

$\overline{x(1)}[1(0),2(1),4(1),21(1)]+x(1)[1(1),23(1),24(1)]$

$x(2)[9(1),22(1),24(1)]$

Under the given set of faults the literal propositions will be reduced to the following set of equations:

Fault#	$y'(1)$	$y'(2)$	z
1	$y(1)+y(2)$	$\overline{x(2)}$	$y(1)$
2	$y(2)+\overline{x(1)}$	$x(1)\overline{x(2)}$	$\overline{x(1)}+x(1)x(2)$
3	$y(2)+\overline{x(1)}y(1)+\overline{y(1)}x(1)x(2)$	$\overline{x(1)}+x(1)\overline{y(1)}$	$\overline{y(1)}x(1)+y(1)\overline{x(1)}+x(1)x(2)$
4	$y(2)+\overline{x(1)}y(1)+\overline{y(1)}x(1)x(2)$	1	$y(1)\overline{x(1)}+x(1)x(2)$

Assuming that the state assignment is A=00, B=01, C=11 and D=10, the flow tables for machines m(0) through m(4) are:

m(0)

y/x	00	01	10	11
A	B,0	B,1	A,0	D,1
B	D,1	A,0	C,1	A,1
C	D,0	D,1	C,1	D,1
D	D,1	C,0	C,0	C,1

m(1)

y/x	00	01	10	11
A	B,0	B,1	A,0	D,1
B	D,1	D,1	C,1	C,1
C	D,0	D,1	C,0	D,1
D	D,1	D,1	C,1	C,1

m(2)

y/x	00	01	10	11
A	D,1	B,1	C,1	D,1
B	D,1	A,0	C,1	A,1
C	D,1	D,1	C,1	D,1
D	D,1	C,0	C,1	C,1

m(3)

y/x	00	01	10	11
A	B,0	B,1	B,0	D,1
B	D,1	A,0	D,1	A,1
C	D,0	D,1	D,0	D,1
D	D,1	C,0	D,1	C,1

m(4) y/x	00	01	10	11
A	B,0	A,1	A,0	C,1
B	D,1	A,0	C,1	A,1
C	D,0	C,1	C,0	C,1
D	C,1	C,0	C,1	C,1

The sequential table can be now found:

Table 2.1 Sequential table for machine M2.2

States of m(0)...m(4)	Inputs			
0 1 2 3 4	00	01	10	11
A A A A A	B B * B B	B B B B A	A A * B A	D D D D C
B B * B B	D D * D D	A * * A A	C C * D C	A C * A A
B B B B A	D D D D *	A * A A *	C C C D *	A C A A C
A A * B A	B B * * B	B B * * A	A A * * A	D D * A C
⋮				
A * * * A	B * * * B	B * * * A	A * * * A	D * * * C
A C * * A	B D * * B	B D * * A	A C * * A	D D * * C

The successor tree can be constructed and the shortest input sequence leading to the state with stars for all detectable faults can be derived, (see Fig. 2.3). The minimal testing sequence is 10,01,01.

Alternatively the sequential table can be rearranged in such a way that all successor states under any one input are written in one column, (see Table 2.2). Under this arrangement it is easier to find the testing sequence, since during the testing experiment, it is only nescessary to pass through an *

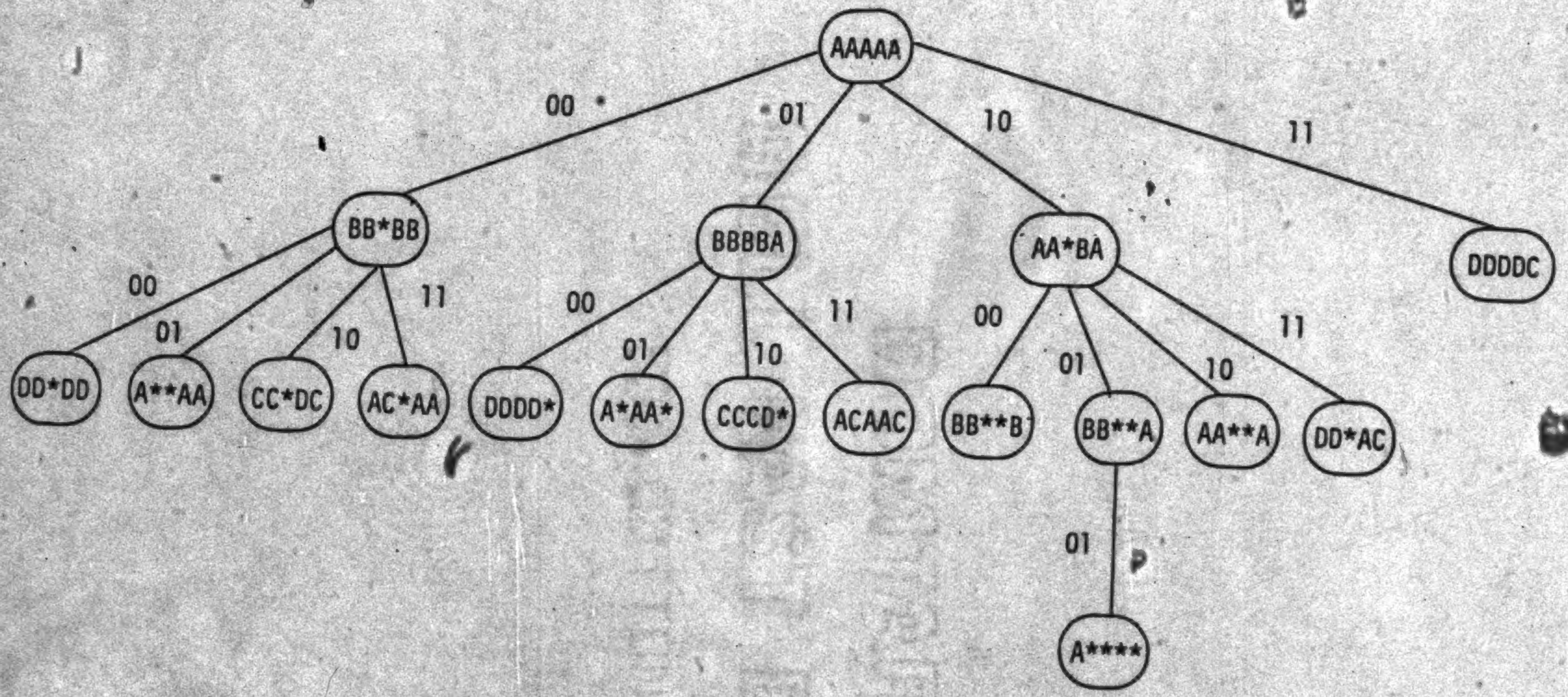

AAAAA
00
01
10
11
BB*BB
BBBBA
AA*BA
DDDDC
00
01
10
11
DD*DD
A**AA
CC*DC
AC*AA
00
01
10
11
DDDD*
A*AA*
CCCD*
ACAAC
00
01
10
11
BB**B
BB**A
AA**A
DD*AC
01
A****

Fig. 2.3 Successor tree for machine M2.2

Table 2.2 Rearranged sequential table for M2.2

Input	States of m(0)...m(4)	1	2	3	4
00	A A A A A		*		
	B B * B B		*		
	B B B B A				*
	A A * B A		*	*	
01	A * * * A	*	*	*	
	A C * * A		*	*	
	A A A A A				
	B B * B B	*	*		
	B B B B A	*			*
	A A * B A		*	*	
10	A * * * A	*	*	*	
	A C * * A		*	*	
	A A A A A		*		
	B B * B B		*		
	B B B B A				*
	A A * B A		*	*	
	A * * * A	*	*	*	
	A C * * A		*	*	
11	A A A A A				
	B B * B B		*		
	B B B B A				
	A A * B A		*		
	A * * * A	*	*	*	
	A C * * A		*	*	

state at least once for each faulty machine. Any of the techniques used for minimization of prime implicant tables [3,9] can be invoked here, and used to minimize the number of states which have to be visited in order to detect all faults. This itself does not guarantee a minimal length testing sequence; however, the sequence is likely to be minimal or near minimal. For instance, in this example, it is possible to cover all faults by following the states BB**A,A**B* and A*A** by inputs which will take the machine through transitions in which faults 1,4,3 and 2 will be detected. Since the machines are initially in state AAAAA which is the same as A*A** the input 00 or 10 will test for fault 2. Hovever only input 10 takes the machine directly to the next state selected above. This input takes the machine to state AA*BA which can be followed by the inputs 00,01 or 10 in order to test for fault 3. Input 10 takes the machine directly to the last state which the machine must visit during the experiment. In this state BB**A under the input 01 faults 1 and 4 will be tested. So the sequence obtained by this method is 10,10,01. This sequence is different than the one obtained previously, but it is of the same length.

The following change in the sequential table development algorithm can reduce the size of the table significantly. Every time the successors of a certain state are found, only the one which detects a maximum number of new faults, is considered as a possible successor. All other successors will be ignored and will not be entered as "old states" into the table. Only if none of the successors detects a new fault, will all successors be used. With this change it is impossible to guarantee the minimality of the testing sequence obtained from a sequential table. Results show, however, that the sequences are still nearly minimal while the amount of computation neccessary to generate the table is significantly reduced.

The modified sequential table for the given machine is shown in Table 2.3.

Table 2.3 Modified sequential table for machine M2.2

Machine	Inputs			
01234	00	01	10	11
AAAAA	BB*BB	BB*BB	AA*BA	DD*DC
BB*BB	DD*DD	A**AA	C**DC	A**AA
A**AA	B**BB	B**BA	A**BA	D**DC
B**BB	D**DD	A**AA	C**DC	A**AA
B**BA	D**D*	A**A*	C**D*	A**A*
A**BA	B****	B****	A****	D****

The shortest sequence obtained by a tree search of the modified table would be 00,01,10,00.

A program which finds a nearly minimal testing sequence for a given machine is described in Appendix I. It follows the Poage-McCluskey procedure; however, a modified sequential table and a simpler table search method is used.

The advantages of the Poage-McCluskey method are:

a) A minimal length testing sequence can be produced.

b) The method is general, since it has virtualy no restrictions except for the restriction on the initial state.

c) It can be easily programmed.

d) The modified version can be used to generate nearly minimal testing sequences for large circuits with relatively large fault sets.

On the other hand there are the following problems:

a) The problem of initial state.

b) The large amount of computation required.

The method assumes the existence of a procedure or hardware arrangement under which the machine can be reset into some known state. It does not have to be the ~~same~~ state for both good and faulty machines m(0),...,m(NF) but it is necessary to know the initial state for every fault. An initial composite state of all machines m(0),...,m(NF) will be the first state entered in a sequential table. While to be able to initialize the machine into some predefined state is a problem, it is hardly a disadvantage when compared to other testing methods, since this problem is always encountered.

Alternatively, it is possible to generalize the procedure and find testing sequences for all possible combinations of initial states. This can be done, in principle, simply by constructing sequential tables starting with every possible composite state and then by making sure that some sequence goes through an asterisk state for each sequential machine in each table at least once. The amount of computation this would require, however, seems prohibitive even if the modified sequential table is used.

The method essentially simulates the operation of the machine under all possible faults. Then the minimal testing sequence is extracted by means of a full tree search. This approach is directly responsible for the large amount of computation required to generate minimal or nearly minimal sequences for even small machines if the original method is followed.

Next will be described a method where the amount of computation required to generate testing sequences is reduced.

2.1.2 Path sensitizing [1,9]

This method extends path sensitizing, as used in the generation of tests for combinational circuits to sequential machines. A sequential machine is considered to consist of a number of identical combinational circuits, with the feedback outputs of one circuit connected to the feedback inputs of the next circuit, etc. When the machine is started, the input is applied to the inputs of the first circuit. In the next time interval, an input is applied to the input of the second circuit, etc. The output of the first combinational circuit is the first output of a sequential circuit since the start of the operation, the output of the second circuit is the next output of the sequential machine etc.

It is assumed that:

1) The circuit diagram is known.

2) Both "good" and faulty machines can be reset to a known initial state.

In detail the method can be summarized as follows: Every sequential machine can be considered to consist of a number of identical combinational networks connected as illustrated in Fig. 2.4. The tests can be obtained in the following steps:

1) Set n=1.

2) Attempt to sensitize a path leading from the location of the fault f(n) to the output z(i), for any i=(1,...,NO).

3) If step 2 is not possible, repeat the procedure for the internal output w(j), for any j=1,...,NB. If this is not possible and if no new state is generated then the fault is undetectable. Otherwise, increment n and return to step 2.

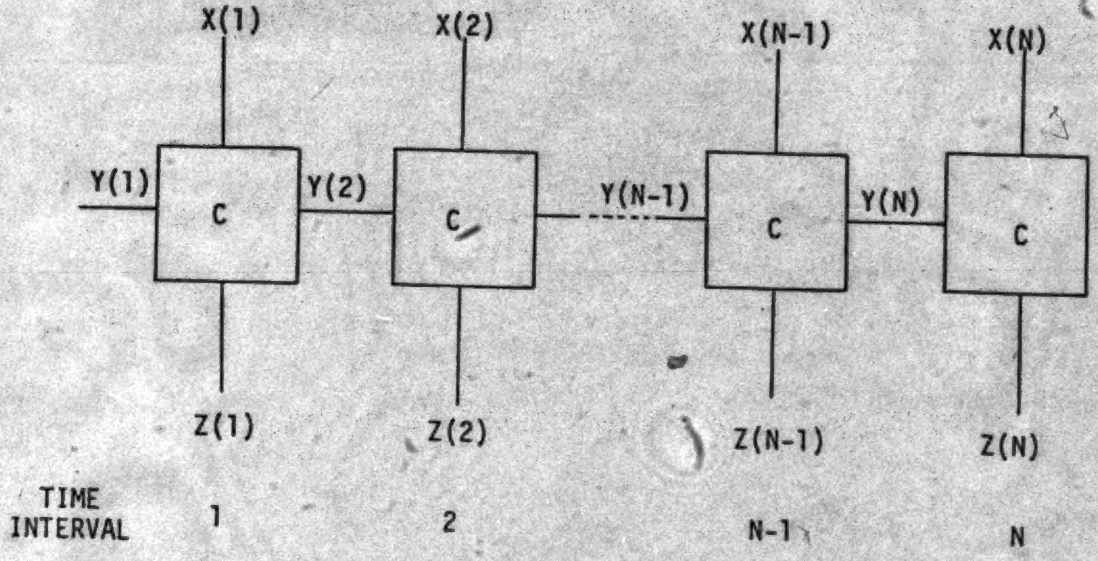

X(1)
X(2)
X(N-1)
X(N)
Y(1)
Y(2)
Y(N-1)
Y(N)
C
C
C
C
Z(1)
Z(2)
Z(N-1)
Z(N)
TIME INTERVAL
1
2
N-1
N

Fig. 2.4 Sequential machine decomposed into a set of combinational ciruits

The sequences obtained for individual faults $f(i)$ can later be rearranged by merging, in order to reduce the total length of the sequence. The path sensitizing method will be illustrated by the following example.

Example 2.3: Find a testing sequence for a s-0 fault at location 1 for the circuit given in Fig. 2.1.

a) Assume that machine is initially in state $y(1)=0$. Then $x(1)=1$ sensitizes the path leading from the fault to $z(1)$. Therefore the sequence 1 is a testing sequence that is valid under the above initial state assumption.

b) Assume that the machine is initially in $y(1)=1$. Then $x(1)=1$ cannot be propagated to $z(1)$ because $y(1)=1$ makes gate 1 insensitive to the state of connection 1. However, $x(1)=1$ will set $y(2)$ to 0 and will be selected as the first member of the testing sequence. With $y(2)=0$ we can set $x(2)=1$ which will sensitize path to $z(2)$. So 11 is a testing sequence for the machine initially in $y(1)=0$.

By merging both subsequences the sequence 1,1 is obtained. This sequence can be used to detect fault $f(1)$ regardless of the initial state of the machine.

In this method the selections of the inputs of the testing sequence are based on local considerations and as the result, only computations for transitions which are actually used in the testing sequence are required. This is an advantage when compared to the previous method where a global approach requires the computation of an outcome of every possible transition in every faulty machine. On the other hand, the local input selection is directly responsible for the fact that the testing sequences obtained by this method are not necessarily of minimal length.

2.2. Machine identification approach [13,14,9]

The characteristic property of testing sequences obtained under this approach is that they distinguish between a good machine and all other machines with the same number or fewer states. Therefore the actual makeup of the machine plus the type and location of faults in the machine are completely immaterial. It is only necessary to know the state table of a machine to develop a testing sequence for the machine. The approach can be summarized as follows:

Assumptions:

a) The state table of the "good" machine m(0) must be known.

b) The machine m(0) must be strongly connected [14].

c) The number of states of the machine m(0) must not increase as result of any fault.

Basic idea: Given a "good" machine m(0) with NS states, NI input symbols and NO output symbols, reffered to as an (NS,NI,NO) machine, it is possible to construct an input sequence, which also will be called a checking experiment, the response to which distinguishes this machine from any other (NS,NI,NO) machine. The basis for this approach was laid by Moore [9] in 1956 who proved that such experiments of length L, where

L<NS**2*(NS*NO)**(NS*NI)

can be found for any machine satisfying the above assumptions.

2.2.1 HENNIE's method [13]

In this method testing sequences are designed in two stages. First, all states of a machine are identified by responses to a distinguishing sequence [14,13]. Then all transitions of the machine are tested by applying all possible inputs to all states and by observing the outputs and responses to distinguishing sequences (DS) which follow every transition test.

The following assumptions are made:

a) The state table of the "good" machine m(0) must be known.

b) The machine m(0) must be strongly connected.

c) The number of states of the machine m(0) must not increase as the result of any fault.

d) The machine m(0) must have a distinguishing sequence.

The detailed description of the method follows:

1) Assuming that there are NS states in the machine, construct a direct sum (composite) [14] table of all state tables having NS or fewer states.

2) Find a preset or adaptive homing sequence [14] which divides all states into sets of equivalent states. If at least one of the states in the final set belongs to the original table, the machine operates correctly. If, on the other hand, none of the states in the final set belongs to the "good" table, that table no longer represents the behaviour of the tested machine, and therefore the machine does not function properly.

Example 2.4: Find a homing sequence which distinguishes between the following two machines m(0) and m(1).

m(0)

S/I	0	1
A	B,0	A,0
B	B,0	C,0
C	A,1	C,0

m(1)

S/I	0	1
E	F,1	E,1
F	E,1	G,0
G	E,1	G,0

As illustrated in Fig. 2.5, the sequence 0,0 divides states of m(0) and m(1) into two groups where there are only states from one table in each group. Subsequently this sequence can be used as a test to decide whether the given machine is m(0) or m(1).

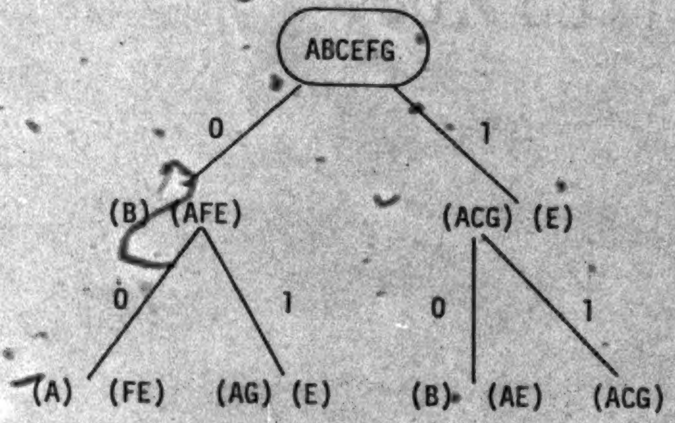


Fig. 2.5 Homing tree for machines m(0) and m(1)

To design checking experiments in the above manner is obviously impractical since the number of tables which satisfies condition of having NS or fewer states is large and grows rather rapidly as NS increases[1].

---

[1] There are (NS*NO)**(NS*NI) possible (NS,NI,NO) machines. Even if all isomorphic machines are disregarded, the number of distinguishable (NS,NI,NO) machines approaches (NS*NO)**(NS*NI)/NS! assymptotically [14].

In practice, the checking experiments are designed in the following way:

1) Find a preset or adaptive homing sequence[2] [14] which identifies the current state of machine m(0) followed by a sequence which takes m(0) to some fixed state S(i).

2) Construct the sequence, which forces m(0) through all its states. Every time a new state would be reached the distinguishing sequence is applied. In this way the responses to the DS by all unique states can be found. It is imposible to make any association between the states as identified by the DS during the experiment and the states of m(0). If Q(i) is the final state resulting from DS applied to the state S(i), and T(Q(i),S(j)) is a transfer sequence to state S(j) then this part of the experiment can be written as:

T(Q(1),S(2)), DS, T(Q(2),S(3)),...,DS, T(Q(n),S(1))

3) Apply an input sequence which checks all transitions in the given table. Again every tested transition is followed by the DS. The output resulting from the transition along with the outputs resulting from the DS permit the determination of the outcome of every transition. The sequence which checks one transition will be called a cell. The cell which checks the transition from state S under input I will be called a C(S,I)

---

[2] It is always possible to find an adaptive homing sequence for any strongly connected machine [13].

cell. A cell can only be applied to the known state. To check the transition from S(i) under the input I it is possible to use

$$T(Q(i)-1,S(j))\ DS\ T(Q(j),S(i))\ I\ DS.$$

The second transfer sequence is a verified transfer sequence from some state Q(j) to S(i). The first transfer sequence must take the machine into such state S(j) that the following DS will leave the machine in the starting state of the second transfer sequence.

The upper bound E on the length of the checking experiment developed according to this method is

$$E<2*NS*(NI+1)*(NS+L),$$

where L is the length of the DS. The above formula is obtained in the following way. The length of each transfer sequence is less than or equal to NS-1. Then the first two parts of the experiment require at most

$$(NS+1)*L+NS*(NS-1)$$

inputs. To check one transition up to

$$2*(NS-1+L)+1$$

inputs may be required. Therefore a (NS,NI,NO) machine requires

$$NS*NI*(2*NS+2*L-1)+(NS+1)*L+NS*(NS-1)$$

inputs which can be written in more compact way as

$$2*NS*(NI+1)*(NS+L).$$

The following example will illustrate the application of

Hennie's method.

Example 2.5: Find a checking experiment for the machine given in the following table:

S/I	0	1
A	A,0	D,0
B	C,1	B,0
C	D,1	A,1
D	B,0	A,1

1) The machine does not have a preset homing sequence and therefore it also does not have synchronizing sequence [ ]. An adaptive sequence must be found. If after an input 1 is applied, an output of 1 is obtained, the machine is in state A, otherwise it is in the state B or D . If the machine was in B or D, then another input 1 will take the machine to either A or B. Exactly which, can be determined from the output. If the output is 1 the machine is in A otherwise it is in B and the sequence 0,1 must be applied to bring the machine finally into A.

2) State identification. A distinguishing sequence for this machine is 0,0 as can be seen in the predecessor tree in Fig. 2.6.

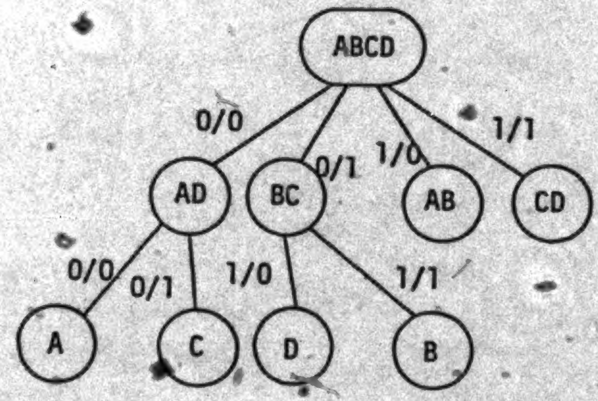

ABCD
0/0
0/1
1/0
1/1
AD
BC
AB
CD
0/0
0/1
1/0
1/1
A
C
D
B

Fig. 2.6 Predecessor tree for machine m(0)

If the machine operates correctly

A followed by 0,0 will take m(0) to A

A " " 1 " " " " D

D " " 0,0 " " " " C

C " " 0,0 " " " " B

B " " 0,0 " " " " D

This leads to the following identification sequence:

Inputs: 001 00 00 00 00

States: a $\underline{d}$ $\underline{c}$ $\underline{b}$ d c[3]

Output m(0): 000 01 10 11 01

The states in the sequence are denoted with the lower case letters since in the experiment the states may not be identical with the states of the "good" machine. It is however convenient to use corresponding lower case letters since the actual states will correspond to states of m(0) if the machine operates correctly. The last state in the sequence must coincide with the starting state of the third part of the experiment. This can be accomplished in this case by appending another distinguishing sequence which is known to take the machine from the state d to state c. If the machine responds to this sequence correctly the machine must have four distinct states.

3) During step 2 the transitions d,0, c,0 and b,0 were checked. The remaining transitions can be tested by the following sequence:

---

[3] Underline indicates that the transition consisting of the underlined state and the following input is tested at that point in the experiment.

Inputs: 1 00 00 0 00 1 00 00 1 00 1 00

States: $\underline{c}$ a a $\underline{a}$ a $\underline{a}$ d c $\underline{b}$ b $\underline{d}$ d

Output m(0):1 00 00 0 00 0 01 10 0 11 1 01

The first three inputs check transition c,1 since the machine is initially in state c. To determine the state of the machine, after this test, another DS is applied. This sequence must take the machine into a, since it has done so before. Now, transition a,0 can be tested by the sequence 0,0,0. If the machine responds correctly it must be stationed in state a after this sequence is applied. Therefore the a,1 transition can be tested by inputs 1,0,0. This particular sequence should leave the machine in state c since it did just that in the second part of the experiment. There are no more c transitions to be tested so the machine is transfered to state b by the input sequence 0,0. This sequence was used and verified in the second part of the experiment. Next cell tests the transition b,1 by applying 1,0,0 which will take the machine to state d (again because the machine responded in the similar manner in the second part of the experiment). Finally, the last cell, tests the d,1 transition.

As was demonstrated in the example the generation of a testing sequence by this method is straigthforward and substantially less tedious then under any circuit testing approach. There is no need to simulate the effect of individual faults on the function of the machine. On the negative side, however, testing sequences are not nearly of minimal length. Also the machine must possess a distinguishing sequence. In the next method this restriction is removed.

2.2.2. HENNIE-HSIEH's method [13,16,9]

The following assumptions are made:

a) The state table of the good machine m(0) must be known.

b) The machine m(0) must be strongly connected.

c) The number of states of the machine m(0) must not increase as a consequence of any fault.

The detailed description follows:

This method was primarily designed for machines without a DS. Hennie showed that the state of any strongly connected machine which is reduced can be determined by a set of characterizing sequences. This set consists of sequences which when applied to the same state can be used to identify it. Hennie also proved [13] that the number of sequences need not exceed* NS-1 and that the length of any single sequence is also less than or equal to NS-1.

---

* Any two states of any reduced machine can be distingished by an input sequence of length NS-1. The first sequence in the set will partition the set of all states into at least two classes according to their responses. The next sequence will partition at least one class to two new classes and there will be at least three classes of states. After NS-1 repetition of the above process all states will be uniquely identified.

The case when the state can be identified by just two characterizing sequences will be described first. Since the characterizing sequences must be applied to the same state it is necessary to find the sequence which leaves the machine in the same state twice. Suppose that X(1) and X(2) constitute a set of characterizing sequences for machine m(0). Let Z1(i) and Z2(i) denote the responses to the sequence X(1),X(2) applied to the machine in state S(i). Also denote the final states after X(1),X(2) has been applied, by P(i)(Q(i)). Finally T(S(i),S(j)) will denote the transfer sequence between states S(i) and S(j). An effect of the sequence[5]

[X(1)T(P(1),S(1))]*(NS+1) X(2)

on machine m(0) currently stationed in state S(1) can now be investigated. Before each application of X(1)T(P(1),S(1)) the circuit must be in a state which responds to X(1) with Z1(1). But there are, at most, NS distinct states which do so. The state after the last application of X(1)T(P(1),S(1)) must be identical to the state of the circuit just before some application of X(1)T(P(1),S(1)). Therefore both characterizing sequences X(1) and X(2) are applied to the same state. It is the state in which the circuit was just before X(2) was applied, and which responds to X(1) with Z1(1) and to X(2) with Z2(1). The sequence

---

[5] S*N indicates that the sequence S is to be repeated N times.

[X(1)T(P(1),S(1))]*(NS-1) X(2)

is then the <u>locating</u> <u>sequence</u> L(S(i)), of the state S(i). Similarily we can find locating sequences for other states.

In the following example a checking experiment for a machine which does not posses a distinguishing sequence will be constructed.

<u>Example</u> <u>2.6</u>: Find a checking experiment for the machine given in the following table:

S/I	0	1
A	B,0	D,0
B	A,0	B,0
C	D,1	A,0
D	D,1	C,0

The machine does not have a DS. The characterizing set is {0,10}. The locating sequences are:

```
L(A): Input :000000000010      L(B):000000000010
      Output:000000000001            000000000000

L(C): Input :010101010110      L(D):0000010
      Output:101010101000            111101
```

In this case the locating sequences can be shortened since they must contain at least one input 0 that causes an output 0 so that the actual circuit with correct behaviour can have at most three states which respond to the input 0 with an output of 1. Therefore the locating sequence for C and D need only contain four repetitions of 01 and 0, respectively. Also the locating sequences for C and D show two distinct states which respond with an output of 1 to a 0 input. Therefore the locating sequences for A and B need only contain three repetitions of 00. Hence, the reduced sequences are:

```
L(A): Input :00000010        L(C):0101010110
      Output:00000001            :1010101000

L(B): Input :00000010         L(D):000010
      Output:00000000             :111101
```

The complete checking experiment becomes: The sequence 010101 can be used to synchronize the machine to c. The problem of state identification can now be solved. The appearance of a given locating sequence and its correct response at several places in an experiment guarantees that the circuit is in the same state at the end of each appearance. It is not known which state it is unless it is possible to bring the machine into the same state twice and apply both characterizing sequences. If the circuit is in the state A and the sequence

L(a) 001L(a) 10

is applied it is possible to determine whether the circuit is in the state D at the end of L(a). Once this is known, the transition from D can be checked. For example the D,0 transition can be checked by the sequence

L(a)0X(1)T(D,A)L(a)0X(2)=L(a)0011L(a)10

The following sequence then forms the complete experiment:

L(c)L(b)L(a)L(d)0L(d)10L(d)00L(d)010L(d)11001L(d)
100L(d)1010L(d)1110L(d)110101L(d)111101100001L(d)
110010110110

The locating sequences should be arranged in such an order that a minimal number of transfers is required.

The design of a checking experiment for machines with more than two characterizing sequences is different from the previous case in two ways:

1)The design of a locating sequence is more complicated. Let X(1),....,X(k) be a set of characteizing sequences for machine m(0). Also let Y(i) denote a sequence which contains X(i) followed by a transfer sequence to state S(i). Consider the effect of

Y(1)*(NS+1) Y(2)

on the circuit. Just before Y(2) is applied, the machine must be in a state which responds to X(1) and X(2) just like state S(i). Now consider the sequence

((Y(1)*(NS+1) Y(2))*(NS+1) Y(1))*(NS+1) Y(3).

The correct response to this sequence indicates that the state of the circuit before the application of Y(3) is one that responds correctly to the first 3 characterizing sequences. The latter sequence must be repeated NS+2 times, substituting Y(l) by Y(l+1) in the last repetition until Y(k) is reached. Similarily the locating sequences for the other states can be obtained.

2) Each transition must be examined k times rather then 2 times.

Hsieh [16,9] simplified the above procedure for machines with at least one simple I/O sequence. The simple I/O sequence can be generated only by the machine in a particular state. It is such that when the sequence I is applied to the machine it will respond with a particular output sequence Z if and only if the machine is in state S(i).

The simple I/O sequences must be validated by a set of characterizing sequences before they can be used in an experiment. In order to do so it is necessary to display NS-1 other states which respond to the I sequence with an output sequence different than Z. To display the response of sequence I to state S(i) the following sequence is applied:

L(S(j)) T(Q(j),S(i)) X(1) L(S(j)) T(Q(j),S(i)) X(2) L(S(j))
T(Q(j),S(i)) I.

The homing sequence can be used instead of locating sequences. Hsieh showed [16,9] that if I is an I/O sequence displaying N distinct states and m(0) is a machine with less than 2*K states that can generate I, then I is a valid homing sequence of m(0). This is true because in the worst case the sequence will respond with an identical response to a pair of states. If there are 2*K-1 states there must be at least one response uniquely associated with one state.

With homing sequence H available, an I/O sequence can be validated as follows:

H,T(S(0),S(j))I,H,T(S(0),S(i))X(1),...,H,T(S(0),S(i))X(p)

A simple I/O sequence which has been verified somewhere in the experiment can be used instead of locating sequences. If I is a verified I/O sequence which leaves the machine in state S(0) it can be used to check the 0 transition from S(0) in the following way:

I,T(S(0),S(i))X(1),I,T(S(0),S(i))X(2),...,IT(S(0),S(i))X(p)
I,T(S(0),S(i))0X(i),I,T(S(0),S(i))0X(2),...,I,T(S(0),S(i))0X(p)

The generation of a testing sequence can be summarized in the following procedure:

1) Find a sufficient set of I/O sequences to verify a simple I/0 sequence.

2) Using a valid I/0 sequence find sequences neccessary to verify all transitions in the given flow table.

3) Use overlap wherever possible.

In the following example, the application of the procedure will be shown.

Example 2.7.: Construct a checking experiment for the machine given in Example 2.6 using Hsieh's method.

It is convenient to convert the state table to its canonical form since it will allow to indicate each transition with an input/output pair along with the state to which the input was applied. By assigning a=0/0, b=0/1 and c=1/0 and by using subscripts to differentiate between identical I/O pairs resulting from different states the following table is obtained:

S/I	0	1
A	B,a(1)	D,c(1)
B	A,a(2)	B,c(2)
C	D,b(1)	A,c(3)
D	D,b(2)	C,c(4)

From the predecessor tree it can be seen that states A and B can be identified by the input sequence 0,1,0. This sequence results in the following output sequences:

A:000 B:001 C and D:101

The machine therefore has two simple I/O sequences

a(1)c(2)a(2) and a(2)c(1)b(2).

It also has a homing sequence

a(1)c(2)a(2)c(1)b(2).

The homing sequence contains both simple I/O sequences and therefore it is only necessary to display states C and D in order to validate this sequence. This can be done with the following sequences:

C: H,c(4)b(1), H,c(4)c(3)a(1), H,c(4)b(1)c(4)b(1) and

D: H,b(2), H,c(4)b(1), H,b(2)c(4)b(1).

Since C and D responds differently to input 0 than A and B they will respond differently to 010 and the last subsequence can be omitted from both sets.

The transitions can now be checked using the following sequences:

A,0 a(1)c(2)a(2) (a(1)c(2)a(2))
(a(1)c(2)a(2))a(1) (a(2)c(1)b(2))

A,1 (a(1)c(2)a(2)) (a(1)c(2)a(2))
(a(1)c(2)a(2))a(1) (a(2)c(1)b(2))

B,0 (a(1)c(2)a(2))a(1) (a(2)c(1)b(2))
(a(1)c(2)a(2))a(1)a(2) (a(1)c(2)a(2))

B,1 (a(1)c(2)a(2))a(1) (a(2)c(1)b(2))
(a(1)c(2)a(2))a(1)c(2) (a(2)c(1)b(2))

C,0 (a(2)c(1)b(2))c(4)b(1)
(a(2)c(1)b(2))c(4)b(1)b(2)
a(2)c(1)b(2))c(4)c(3)a(1)
(a(2)c(1)b(2))c(4)b(1)c(4)b(1)

C,1 (a (2) c (1) b (2) c (4) c (3) a (1)
(a (2) c (1) b (2)) c (4) c (3) (a (1) c (2) a (2))
D,0 (a (1) c (1) b (2)) b (2) , (a (2) c (1) b (2)) c (4) b (1))
(a (2) c (1) b (2)) b (2) b (2) , (a (2) c (1) b (2)) b (2) c (4) b (1)
D,1 (a (1) c (1) b (2) b (2) (a (2) c (1) b (2) c (4) b (1)
(a (2) c (1) b (2)) c (4) b (1) , (a (2) c (1) b (2)) c (4) c (3) a (1)

By eliminating repetitions and sequences contained in other sequences and and by merging of the sequences together the following sequences are obtained

1. a (1) c (2) a (2) a (1) a (2) c (1) b (2)
2. a (1) c (2) a (2) c (1) c (4) b (1)
3. a (1) c (2) a (2) a (1) a (2) a (1) c (2) a (2)
4. a (1) c (2) a (2) a (1) c (2) a (2) c (1) b (2)
5. a (2) c (1) b (2) c (4) b (1) b (2)
6. a (2) c (1) b (2) c (4) b (1) c (4) b (1)
7. a (2) c (1) b (2) c (4) c (3) a (1) c (2) a (2)
8. a (2) c (1) b (2) b (2) b (2)
9. a (2) c (1) b (2) b (2) c (4) b (1)

The sequences which check simple I/O sequences may be included here:

10. a (1) c (2) a (2) c (1) b (2) b (2)
11. a (1) c (2) a (2) c (1) b (2) c (4) c (3) a (1)
12. a (1) c (2) a (2) c (1) b4c (4) b (1)

By merging these sequences the following shorter sequences are obtained:

4-11-7-2: a(1)c(2)a(2)a(1)c(2)a(2)c(1)
b(2)c(4)c(3)a(1)c(2)a(2)c(1)c(4)b(1) = X(1)

3-12-6 : a(1)c(2)a(2)a(1)a(2)a(1)c(2)a(2)c(1)
b(2)c(4)c(1)c(4)b(1) = X(2)

1-5 : a(1)c(2)a(2)a(1)a(2)c(1)
b(2)c(4)b(1)b(2) = X(3)

10-8 : a(1)c(2)a(2)c(1)b(2)b(2)b(2) = X(4)

and 9 is X(5).

All of these sequences end with b(1) or b(2) in final state D. They start in either A or B. The transfer sequences between D and A or B are c(4)c(3) or c(4)c(3)c(1) respectively. All sequences can now be joined into the following checking experiment:

X(1)c(4)c(3)X(2)c(4)c(3)X(3)c(4)c(3)X(4)c(4)c(3)a(1)X(5)

The length of the sequence is 62, which is a considerable saving from 132 in Example 2.6.

It should be apparent even from the previous example that to generate the sequences by the above method is involved, and also that the testing sequence tends to be rather long.

It was discovered that if response to a certain transition tests is correct, there is no need to perform tests on some other transitions. The following method identifies such transitions.

2.2.3. GONENC's method [11]

The main contribution of this paper is the algorithm which allows the construction of checking experiments of optimal length. This algorithm helps to determine in which order individual transitions should be tested so that the total length of the experiment is close to minimal.

The following assumptions are made:

a) The state table of the "good" machine m(0) must be known.

b) m(0) must be strongly connected.

c) The number of states of m(0) must not increase as the result of any fault.

d) The machine must posses a distinguishing sequence, DS.

The following definitions will be useful in the description of the method. The source state is a state which is not the terminal state of DS applied to any state of the machine. The recognized state is a state which can be identified with the state of the given flow table.

The state can be recognized in one of the following ways:

a) An application of a DS. Then the state is D-recognized.

b) An application of the homing sequence. Then the state is Q-recognized.

c) An application of a sequence which is known to force the machine into a certain state.

The detailed description of the method can now be given. Disregarding synchronization, the testing sequence consits of two parts: In the first part, all states are associated with their responses to a DS, as was the case in Hennie's method. This time, however, it is also necessary to find all Q-states which are the terminal states after application of a DS to all states of the tested machine.

The sequence can be obtained in the following steps:

1) Find the shortest DS.

2) Find all source states.

3) Choose one of the source states as an initial state. If there is no soufce state start with any state.

4) Apply the DS.

5) If a state which was not recognized is entered, return to 4. (This guarantees recognition of all Q-states.) Otherwise go to 6.

6) Apply the DS.

7) If there is still at least one source state which was not recognized, go to 8. Otherwise go to 9.

8) Apply a transfer sequence T which brings the machine to another source state and go to 4.

9) If there is still at least one state which was not recognized go to 10, otherwise stop.

10) Apply transfer sequence T to the state which is still to be recognized and return to 4.

During the second part of the experiment individual transitions are checked. The cell which is used to check the transition from the state S under input I will be called an SI cell and consists of input I followed immediately by the DS. The cell can only be applied to the recognized state. Every cell leaves the machine again in a recognized state because all D-states were identified during the first part of the experiment. This part of the checking experiment is best found by constructing the graph, refered to as T-diagram, which contains all the states connected by arcs for each transition accomplished by each cell. Before the procedure for the generation of optimal sequence can be given, the following definitions must be made: A path is a set of successive arcs in the T-diagram. A simple path is a path where no arc is used more than once. A covering is partitioning of all arcs of the graph into simple paths which are disjoint. If O is a number of arcs leaving a state and I is number of arcs entering a state then

if O=I the state is an R type state,

if O>I then the state is an F type state,

if O<I then the state is a P type state.

If I=O for all states of the T-diagram the graph will be called Eulerian. In this case the set R contains all states of the machine and the sets F and P are empty. For a connected Eulerian graph there exists a unique minimal covering[6]. If the graph is not connected it is necessary to treat every section separately. If the graph is not Eulerian then every minimal covering of the i-th section consists of k paths

$$k = (O(i)-I(i)) = (I(i)-O(i))$$

each of which joins a vertex in the set F with a vertex in the set P. For non-connected graphs it is again necessary consider each section separately.

The algoritm for construction of the second part of the checking experiment follows:

1) Start from any state in set F; however, it may be necessary to start in the final state of the first sequence, even if the state is not in the set F.

---

[6] The connected Eulerian graph forms the circular finite path where the initial vertex coincides with the terminal vertex.

2) Each time an arc is followed, erase it from the T-graph.

3) Do not use a arc which is an bridge at that particular moment if any other arc can be fcllowed. A bridge is an arc that would divide the graph into two components it it was deleted.

4) When it is not possible to continue, transfer the machine to some other state in the set F in such a way that eventually there will be l(j)=O(j)-I(j) starts made from each state.

The following example will illustrate the application of this method.

Example 2.8: Find an optimal testing sequence for machine m(0) given by the following flow table:

S/I	0	1
1	2,0	6,0
2	3,0	5,1
3	4,1	2,0
4	5,0	1,0
5	6,0	4,1
6	2,1	5,1

A sequence 0,0,1 is the shortest DS. States 3 and 6 are the source states. Choose 3 as a starting state. The machine has SS which is 11011100 for 3. The first part of the sequence then is:

Inputs: 001 001 001 001 0 001 001 001 001
States: 3 4 5 5 5 6 2 1 2 1
Output: 101 001 011 011 0 100 010 000 010

Now F and P sets can be obtained obtained: F={3,4,6}, P={1,2,5}, see T diagram in 2.7. Then k=13+14+16=5. So the sequence will have to cover 5 paths if it can start in 3,4 or 6. It will have to contain 6 paths and k-1=5 transfers if it starts in other state. Since the terminal state of the first part of the sequence is 1 the following covering is obtained:

11255242-35-45-65-61-31.

In the above sequence, the digits indicate states visited and dashes indicate transfers. Using the symbol "D" to indicate all inputs of the distinguishing sequence, the corresponding input sequence is:

0D1D1D1D0D0D1D00D10D01D00D001D.

Some of the cells can be eliminated from the experiment Assume that X is an input sequence consisting of 2 parts X(1) and X(2). Also assume that S denotes the next state function and Z is an output function. Then if

$S(S(i),X(1))$, $Z(S(i),X(1))$ and
$S(S(i),X)$, $Z(S(i),X)$

are known, the information about

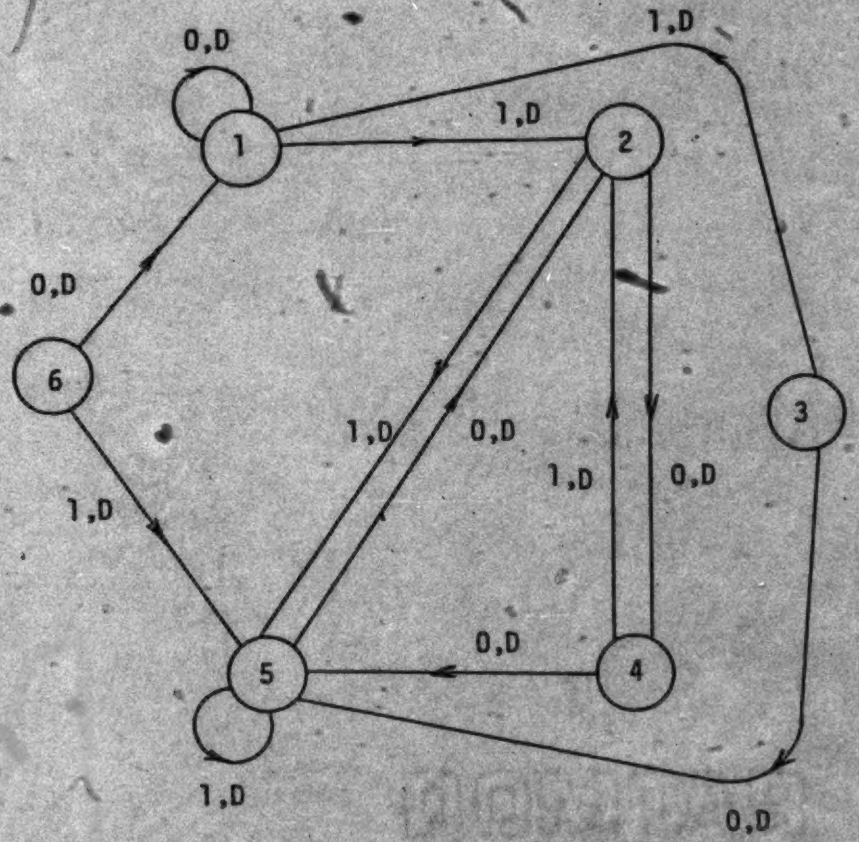

0,D
1,D
1,D
1
2
0,D
6
3
1,D
0,D
1,D
0,D
1,D
0,D
5
4
1,D
0,D

Fig. 2.7 T-diagram

S(S(i),X(2)) and Z(S(i),X(2))

can be deduced. In this example DS consists of the two parts, 00 and 1. Then all 1 transitions from the states reached under 00 can be eliminated from the experiment provided that the 0 transitions are checked before any eliminated transition is used in the sequence. From the reduced T diagram given in Fig. 2.7 the following covering is produced:

11245-61-35

which results in the input sequence

OD1DODODCODOOOD.

This method concludes discussion of the circuit testing and the machine identification approaches. In the following method an attempt was made to combine the advantages of the two above approaches with the hope that some of the weaknesses of the individual approaches can be overcome.

2.3 Hybrid approach.

In this approach Hennie's method for generation of testing sequences is used to identify individual states of a machine and to generate cells which can be used to test individual transitions. Not all of these cells are actually used in the experiment. The effects of a given set of faults on a circuit are first analyzed and the transitions which cannot be changed are not tested. This results in shorter testing sequences.

2.3.1 KOHAVI's method [19]

The following assumptions are made:

a) The machine circuit diagram is known.

b) The state table of the machine m(0) must be known.

c) m(0) must be strongly connected.

d) The number of states of m(0) must not increase as the result of any fault.

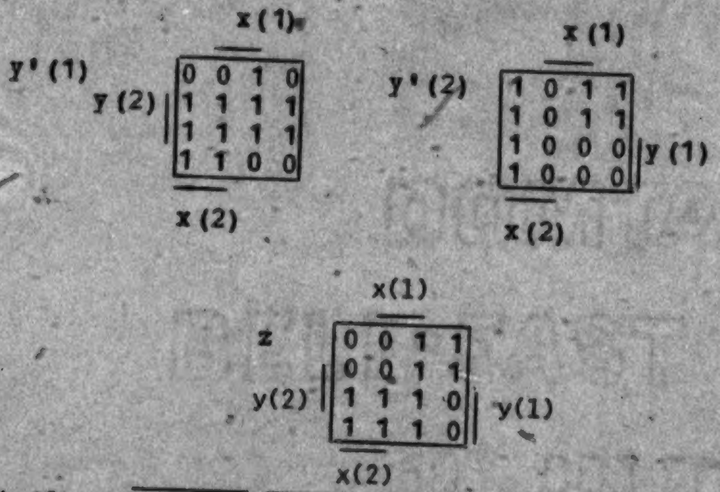
The description of the method follows: First the state table corresponding to the given machine m(0) is found. From the circuit diagram identify the transitions which would be influenced by a given set of faults. Only these transitions have to be checked. Further steps are identical with those described in sections 2.1.1. or 2.1.2. The method is illustrated in the following example.

Example 2.9: Find the sequence which detects all single stuck type faults in the machine given in Fig. 2.2. The maps given in Table 2.5 describe the operation of the circuit.

The s-0 tests are input combinations which set the outputs of the tested AND gate to 1 while no other AND gate is 1 [19]. They can be derived directly from the maps.

$$T(y(1))=\{\overline{2,6};\overline{1,5,9,11,15};\overline{12}\}$$

Table 2.4 Maps of machine m(0)



y'(1)

0	0	1	0
1	1	1	1
1	1	1	1
1	1	0	0

y'(2)

1	0	1	1
1	0	1	1
1	0	0	0
1	0	0	0

z

0	0	1	1
0	0	1	1
1	1	1	0
1	1	1	0

$T(y(2))=\{\overline{0,1,2,3};\overline{8,9,12,13}\}$

$T(z)=\{\overline{8,9};\overline{14,15};\overline{3,6,7}\}$

All tests for one fault are grouped togeher in the above sets. The groups are separated by the semicolons. At least one test from each group must be selected for the testing. The selection can be done by any of the methods used for minimization of the prime implicant tables.

Since a s-1 fault has the same effect as adding one prime implicant to the function, s-1 tests are the input combinations adjacent to the inputs which set the tested AND gate to 1 normally. Naturally no input combinations which would result in a 1 output of the correct network can be used [19]. The fcllowing are the s-1 tests again obtained directly from the maps.

$T(y(1))=\{4;8;14\}$

$T(y(2))=\{\overline{4,5};\overline{10,11}\}$

$T(z)=\{\overline{10,11};\overline{4,5}\}$

Finall set of tests will be:

$T=\{4,8,14,12,11,2\}$

The maps of the circuit can be combined into a following state table, where testing transitions will be marked by *.

S/I	0	1	3	2
00=A	$B,0^{0}$	$A{*}0^{4}$	$C{*}1^{12}$	$B{*}1^{8}$
01=B	$C,0^{1}$	$D,0^{5}$	$C,1^{13}$	$C,1^{9}$
11=C	$C,1^{3}$	$D,1^{7}$	$D,1^{15}$	$D{*}0^{11}$
10=D	$C{*}1^{2}$	$D,1^{6}$	$A{*}1^{14}$	$A,0^{10}$

The circuit has a synchronizing sequence 0,0 and a distinguishing sequence 2,2. A testing sequence can be constructed as follows:

1) State identification:

X:0 0 2 2 2 2 2

S: $c^{11}d\ a^{8}b$

Z: 0 0 1 1 0

2) Transition tests:

The transitions C,2 and A,2 were already tested during the first part of the experiment. The remaining * transitions can be tested by the following sequence:

X: 2 2 2 3 2 2 1 2 2 2 3 2 2 2 0 2 2

S: $b\ c\ d\ a^{12}c\ d\ a^{4}a\ b\ c\ d^{14}a\ b\ c\ d^{2}c$

Z: 1 0 0 1 0 0 0 1 1 0 1 1 1 0 1 0 0

This example concludes the survey of previous results. In summary it can be said that while the circuit testing approach is very general and laborous the machine identification approach is relatively straightforward, but is restricted to a certain class of machines. With the circuit testing approach it is possible to obtain testing sequences of minimal length, or at least sequences which are nearly minimal. Quite opposite is true about the machine identification approach. On top of that, to use the machine identification methods on machines which do not posses a distinguishing sequence is rather cumbersome. By combining the machine identification and the circuit testing approaches into a hybrid approach, the length of the testing sequence is somewhat reduced while retaining the easy generation procedure from Hennie's machine identification method. The length of the sequence, however, is still not nearly minimal and naturally all restrictions accompanying Hennie's method remain.

It can be concluded, therefore, that all current methods are unsatisfactory from one point or another. In the following chapter a testing method which is quite general and which generates testing sequences of nearly minimal length with an acceptable level of effort will be described.

Chapter 3

TESTING SEQUENCES

In this chapter a new method for the generation of testing sequences will be presented. Testing sequences will be developed by forcing a machine into a fault-sensitive situation and examining every possible outcome under any possible fault. Faults will be classified according to their effect on the behaviour of the sequential circuit. The properties of many faults allow the testing procedures to be simplified and, in many cases, the length of the testing experiments to be reduced. Procedures for the generation of testing sequences for machines with a particular type of fault will then be described.

3.1 Fault Classification

A sequential machine m(0) consists of a combinational circuit c(0) with some of its outputs connected via feedback lines to some of its inputs, e.g., see Fig. 3.1. Outputs which are connected to the feedback lines will be called feedback outputs and denoted by W. Outputs of c(0) which are also outputs[7] of m(0) will be denoted by Z. Inputs which are connected to the feedback lines will be called feedback inputs and denoted by Y, while other inputs will be called X.

A logical fault f(i), for all i, in a sequential circuit c(0), which changes machine m(0) into machine m(i), can influence one or more outputs Z, and/or one or more feedback outputs W. Such a fault can be detected by applying tests

t(i,1),...,t(i,NT(i)), where

t(i,j) is the j-th test for f(i), and NT(i) is total number of different tests for f(i).

Assuming that m(0) has NI inputs and NB feedback inputs, each test t(i,j) will consist of NI+NB binary values, NI of which will be applied to the inputs x(1),...,x(NI), and the remaining NB values will be present on the feedback inputs y(1),...,y(NB). The values applied to the input are denoted by U and the values applied to the feedback inputs by S. Thus

t(i,j)={U(i,j);S(i,j)}

={u(i,j,1),...,u(i,j,NI),s(i,j,1),...,s(i,j,NB)}

In other words each test t(i,j) consists of two components:

1) Input test U(i,j), an input which must be applied when t(i,j) is executed.

---

[7] An output of the combinational circuit c(0) can act both as an output Z of the sequential circuit and a feedback output W of the same sequential circuit.

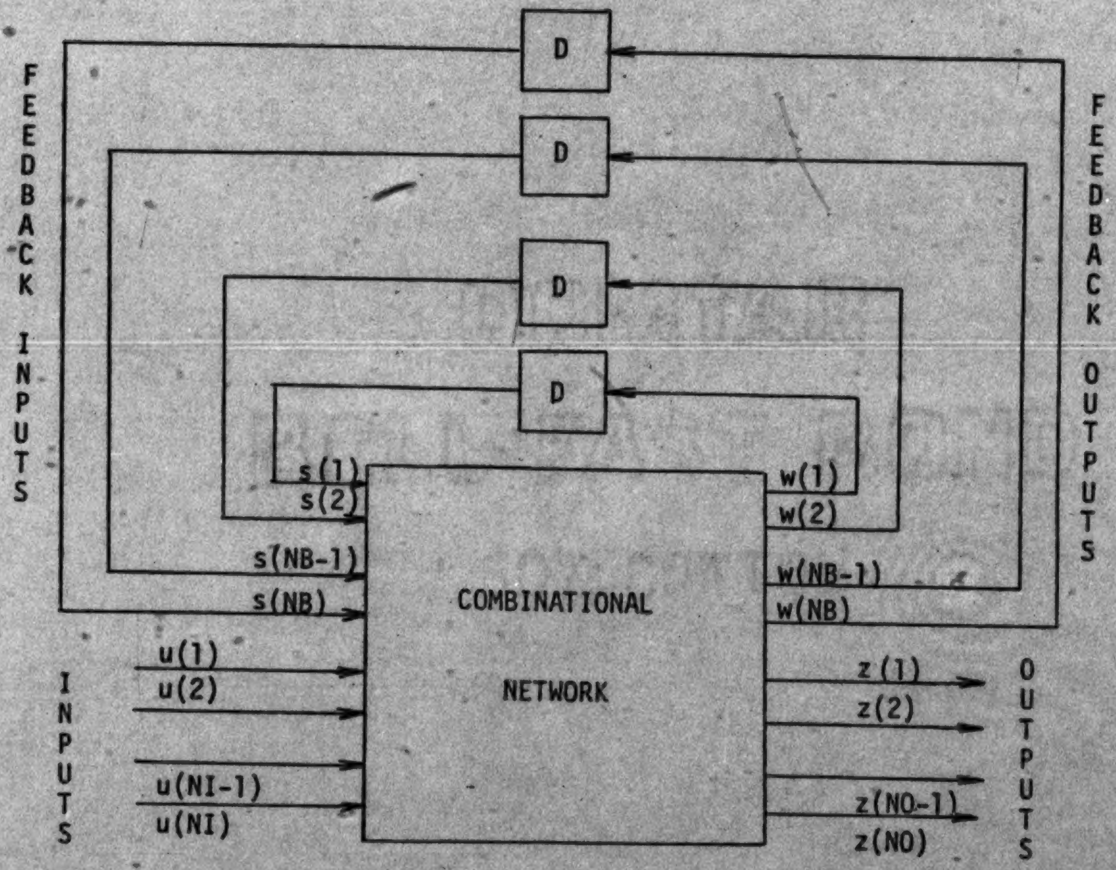

FEEDBACK INPUTS
FEEDBACK OUTPUTS
D
D
D
D
s(1)
s(2)
s(NB-1)
s(NB)
w(1)
w(2)
w(NB-1)
w(NB)
COMBINATIONAL NETWORK
INPUTS
u(1)
u(2)
u(NI-1)
u(NI)
OUTPUTS
z(1)
z(2)
z(NO-1)
z(NO)

Figure 3.1 A sequential machine

2) State test S(i,j), a state which must be present on feedback inputs when t(i,j) is executed.

Because faults in the feedback lines will effect only feedback inputs of a circuit, they will be equivalent to the faults in the feedback inputs Y. Tests T can be obtained by using any method for the generation of tests for a combinational circuit with inputs

$$x(1),\dots,x(NI),y(1),\dots,y(NB),$$

and outputs

$$z(1),\dots,z(NZ),w(1),\dots,w(NB).$$

Definition 3.1: A test t(i,j) is a combination of inputs U(i,j) and feedback inputs S(i,j). When U(i,j) is present on the inputs X and S(i,j) is applied to the feedback inputs Y, the tested machine responds with Z(0),W(0) if it operates correctly, and with $Z(i),W(i) \neq Z(0),W(0)$ if fault f(i) is present.

As a result, a testing sequence E for a set of faults F will consist of a set of tests that contains at least one test for each fault in F. The transition which is executed when the test t(i,j) is applied, will be called $S(i,j),U(i,j) \rightarrow S'(i,j)$ or simply the S(i,j),U(i,j) transition. The state entered by m(0) will be denoted by S'(0,j) or simply by S'(0), while the state entered by m(i) will be denoted by S'(i,j) or by S'(i). Because S'(i) cannot, in general, be examined directly, each test may have to be followed by a sequence, that will propagate a distinction between S'(0) and S'(i) to the outputs Z. Such a sequence will be called a propagating sequence and denoted by D(i). A test t(i), followed by D(0,i), will form one cell c(i) of the testing sequence, which will be sensitive to the fault f(i). The propagating sequence must only be applied if the outputs of the S(i),U(i,j) transition are the same for both m(0) and m(i). If the propagating sequence does not leave m(0) in a state which corresponds to one of the state tests that remain, a transfer sequence, which will take the machine into a new state test S(i,j), must be applied.

These general comments are formalized by the following definitions:

Definition 3.2: A testing sequence or testing experiment E for faults f(1),...,f(NF) is an input sequence which distinguishes machine m(0) from any machine m(1),...,m(NF) into which m(0) transforms itself as a result of f(1),...,f(NF).

Definition 3.3: A propagating sequence D(i) distinguishes the state S'(i) from the state S'(0).

Definition 3.4: A test t(i) followed by a propagating sequence D(i), forms one cell c(i) of a testing sequence.

Definition 3.5: **A transfer sequence** J(S(x),S(i)) is a sequence, which will take m(0) from the state S(x) to the state S(i), from which the next test t(i) can be applied.

From the definition of test and testing sequence, the following lemma is obtained:

Lemma: A fault f(i), for all i, is undetectable if:

1) A test t(i) does not exist for f(i).

2) Fault f(i) in the feedback line(s) does not affect any feedback input.

3) A propagating sequence D(i) does not exist for any test in T(i).

A set of tests

t(1,1),...,t(1,NT(f(1))),...,t(NF,1),...,t(NF,NT(f(NF))),

which detects faults f(1),...,f(NF) in a combinational circuit c(0), can be arranged into a matrix. The arrangement will be such, that the i-th row of the matrix will contain all tests t(i,j) for the fault f(i). It follows that, in order to test for f(i), at least one test must be selected from the i-th row of the matrix. Therefore to test for all faults f(1),...,f(NF), at least one test from each row of the matrix must be selected. It must be noted that the number of columns in the matrix will correspond to the maximum number of tests for fault f(i), i=1,...,NF, and that entries to fill up rows do not constitute legitimate tests.

To perform a test, the input test U(i,j) must be applied to the inputs x(1),...,x(NI) when the machine is in state S(i,j). Therefore the feedback inputs y(1),...,y(NB) must be S(i,j). The next state function of the machine m(0) is known and if the transitions used in testing for the faults f(1),...,f(NF) are governed by the original next state function, the input sequence which takes the machine through all necessary tests, can be readily found. If, however, the transitions used in testing have states altered by the faults f(1),...,f(NF), finding the sequence, which takes the machine through all tests, is more difficult. It is therefore useful to distinguish between the first and second types of fault. These faults will be called output and state faults, respectively. The definitions and procedures for the development of testing sequences for machines with output and state faults can be found in Sections 3.2 thru 3.4.

A single fault may cause an output fault as well as a state fault. In other words, the presence of a fault can alter both the output and the state of the circuit. Two possibilities exist:

1) The fault behaves as an output fault if one test t(i,j) is used, but it behaves as a state fault if another test t(i,k)

is used. Therefore depending on the test used, the fault is either an output fault or a state fault.

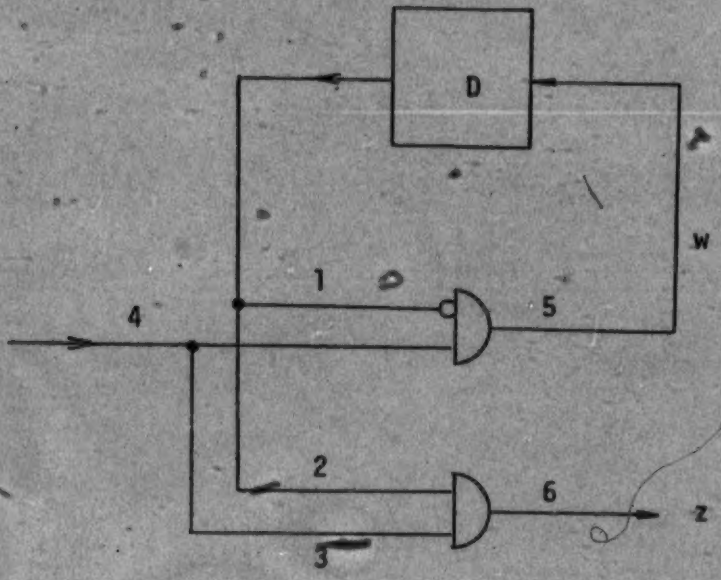
2) The fault behaves as both an output and a state fault under one test t(i,j). Then the fault can be considered as either, but generally it is advantageous to treat it as an output fault.

In the following example faults will classified according to the previous definitions. The faults were arbitrarily selected, to cover all possible types.

Example 3.1: For the machine given in the Fig. 3.2, consider the following set of faults:

Fault #	Location	Type
1	1	s-0
2	2	s-1
3	3	s-1
4	4	s-1
5	5	s-0
6	6	s-0

The state tables of good and faulty machines are shown in Table 3.1.



a)

	x	0	1
u	s		
0	A	A,0	B,0
1	B	A,0	A,1

b)

Fig. 3.2 Circuit diagram and state table of machine M3.1

Table 3.1 State tables of the machines m(0),...,m(6).

	m(0)		m(1)		m(2)		m(3)		m(4)		m(5)		m(6)	
	0	1	0	1	0	1	0	1	0	1	0	1	0	1
A	A,0	B,0	A,0	B,0	A,0	B,1	A,0	B,0	B,0	B,0	A,0	A,0	A,0	B,0
B	A,0	A,1	A,0	B,1	A,0	A,1	A,1	A,1	A,1	A,1	A,0	A,1	A,0	A,0

In the state tables of machines m(2), m(3) and m(6) only the outputs differ from the state table of m(0). Therefore faults f(2), f(3) and f(6) are output faults. On the other hand, only the states of m(1) and m(5) differ from the states of m(0). Therefore f(1) and f(5) are state faults. For machine m(4) in some transitions an output differs from the output of m(0) while in other transitions a state in m(4) differs from m(0). Therefore the fault f(4) can be either a state or an output fault, depending on the test used. If this fault is present, the state resulting from the A,0 transition is different between m(0) and m(4). Therefore if this transition is used for testing, f(4) will behave as a state fault. On the other hand, in the B,0 transition, the outputs of m(0) and m(4) differ, and so if this transition is used as a test for f(4); it will behave as an output fault.

During the construction of a testing experiment E, it is always assumed that only one fault is present at any given time.

This assumption should not be confused with the classical one-fault assumption, which is often used in procedures for the generation of tests fcr combinational circuits. In this and the fcllowing chapters one fault is assumed to mean any logical fault which alters the behaviour of the circuit in some unique way, i.e., m(0) is transfered into m(i). Furthermore such a fault may be a multiple fault in the combinational circuit of the tested machine. Therefore the only difference in developing testing sequences acccunting for single or multiple faults in the combinational part of the machine will be in sets of tests T. It is important to realize that the above assumption is by nc means restrictive, since any physical fault, even though it may cause a multiple stuck type fault, can be included in F as a single fault.

In the next section, testing sequences for machines with output faults will be discussed.

3.2 Testing for output faults

In this section a procedure which can be used to generate the testing sequence for cutput faults, will be presented. The testing sequence will be generated under the assumption that no state faults are present. The procedure can also be used to generate, in succession, the individual cells c(i) which can be used to test for individual output faults f(i).

Definition 3.6: A fault f(i) is an output fault under test t(i,j), if when test t(i,j) is applied and f(i) is present, the output of m(i) differs from the output of m(0).

From the above definition it follows that output faults will alter the outputs of a machine m(0). They may also effect the states of m(0) in the same transitions in which outputs are altered. Assuming, then, that the next state function and the output function of the tested machine along with the set of output faults f(1),...,f(NF) are given, the testing sequence can be constructed by using the following procedure.

Procedure 3.1:

1) Set i to any number between 1 and NF.

2) Using any method for the generation of tests in combinational circuits find at least one test t(i,j) for the fault f(i) in the combinational portion of the tested machine.

3) Find a transfer sequence J(s(p),s(i)), forcing the machine from the current state s(p) to a state test S(i,j).

4) Include J(s(p),s(i)) and the input test U(i,j) in the testing sequence.

5) Change i to a value between 1 and NF, which has not yet been used, then replace s(p) with the successor of s(i),U(i,j) transition, and return to 2. If all values Have been used, then stop.

The order in which the faults are tested can be arbitrary. This means that i can be initialized to any value between 1 and NF and then changed so that all values between 1 and NF are covered. It should be realized, however, that the ordering can effect the number of inputs necessary for transfer sequences and therefore the length of the testing sequence. An algorithm which can be used to order a given set of tests is described in [11]. The only other factor which can have an effect on the length of the testing experiment in this case is the selection of a test from the set t(i,1),...,t(i,NT(f(i))) of tests available for the testing of each fault f(i). There are numerous covering algorithms [3,9] which will yield a minimal set of tests for a given set of faults. Using the best ordering of the minimal set of tests for faults in the combinational part of the machine m(0) does not, however, guarantee the minimal overall length of the testing sequence. There might always be a set of tests which, although not minimal, can be ordered in such a way that the overall length of the sequence is minimal. It is not clear how the above procedure can be used to obtain the experiment with minimal overall length. If only cells testing for individual output are desired, so that they can be reordered or used later on in conjunction with cells for state faults, then only step 2 is executed for each fault f(i).

The following example will illustrate the application of the above procedure.

Example 3.2: Find a testing sequence for machine M3.2 given in Fig. 3.3, assuming the following arbitrary set of output faults:

Fault #	Location	Type
1	23	s-0
2	23	s-1
3	21	s-0

The following tests t(1),t(2),t(3) for faults f(1),f(3),f(3) are obtained.

Tests:	y(1)	y(2)	x(1)	x(2)
t(1)	1	-	1	1
t(2)	1	-	1	0
t(3)	1	-	0	-

These tests, converted into state tests and input tests, are:

t(1): S(1),U(1)=1-,11

t(2): S(2),U(2)=1-,10

t(3): S(3),U(3)=1-,0-

Assume that states '00','01','11' and '10' are named A,B,C and D and that m(0),...,m(4) are reset into state A. Since the tests include only states C or D, a transfer sequence to reach states C or D is required. Input 3 will take the machine from A to C. t(3) can now be applied, setting u(3,2) arbitrarily to 0. The machine will stay in C from which also t(2) can be applied.

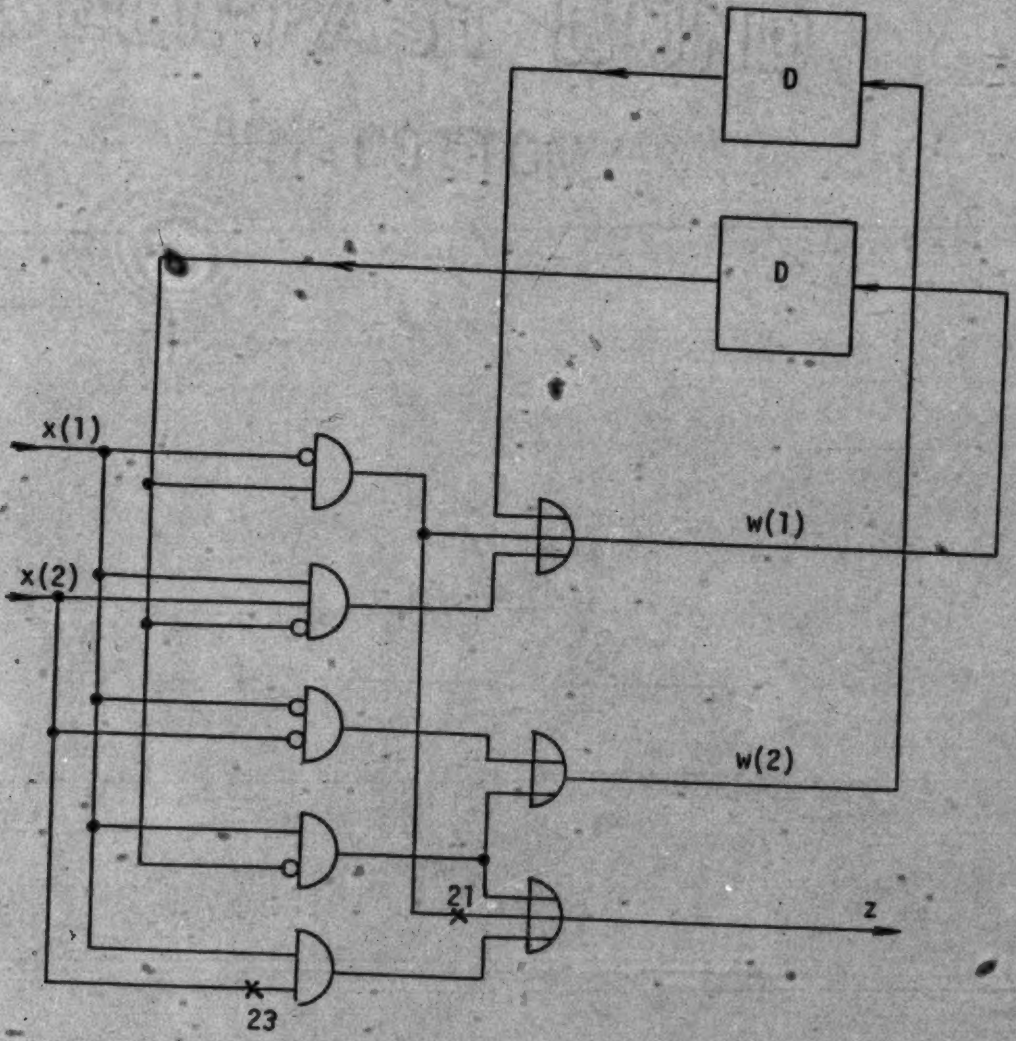

D
D
x(1)
x(2)
w(1)
w(2)
21
23
z

Figure 3.3 Circuit diagram of machine M3.2

Table 3.2 State table of machine M3.2.

S/I	0	1	3	2
00=A	B,0	A,0	C,1	B,1
01=B	C,0	D,0	C,1	C,1
11=C	C,1	D,1	D,1	D,0
10=D	C,1	D,1	A,1	A,0

If the machine operates correctly, this test will take it to D from which t(1) can be applied. A state propagating sequence is not necessary in this case, since only output faults are considered. The testing sequence therefore is:

Input : 3 0 2 3
State : A C C D A
Output m(0): 1 1 0 1
Output m(1): 1 1 0 0
Output m(2): 1 1 1 1
Output m(3): 1 0 0 1

The above testing sequence not only detects all above faults, but it also distinguishes between them.

State faults and methods for the generation of testing sequences for sequential machines with state faults will be discussed in subsequent sections.

3.3 Testing sequences for machines with non-overlapping state faults

In this section a testing procedure which can be used to generate testing sequence for a restricted set of state faults will be developed. This procedure can be also used to generate, in succession, individual cells of the testing sequence which can be used to test for particular state faults.

Definition 3.7: A fault f(i) is a state fault under test t(i,j), if when t(i,j) is applied and f(i) is present, only the feedback output of m(0) is changed, i.e., the next state.

Definition 3.8: A set of state faults F={f(1),...,f(NF)} will be called overlapping if at least one fault f(i) contained in F, influences the state of the same transition as some other fault in F.

Sets of state faults which are non-overlapping have the property that no transition of the tested machine can be changed by more than one fault in the set. In this case, the solution to the problem of the generation of testing sequences is simpler and therefore will be dealt with first. In addition, for non-overlapping state faults, the state sequences of the good and faulty machines resulting from any input sequence will be identical until the test for the fault which happens to be present is applied. Therefore no transition which has not been previously tested can be a member of a propagating sequence or a transfer sequence. Also, a test for any transition must be followed by an appropriate propagating sequence. It should be obvious that the same transition can appear at one part of the testing sequence, while it cannot appear in another part. Therefore it depends on the "current point" in the sequence whether a transition can be used or not. A transition cannot be a member of the testing sequence at its current point unless all of the following conditions are satisfied:

1) It is intended to test the transition at the current point in the experiment.

2) All inputs of the cell testing the transition were applied.

3) The transition must not be influenced by any of the faults f(1),...,f(NF).

In some cases these restrictions can be satisfied by the following procedure:

Procedure 3.7:

1) If faults f(1),...,f(NF) are non-overlapping, then set i to any value between 1 and NF.

2) Find at least one test t(i,j) for the fault f(i).

3) Find and apply the transfer sequence J(s(p),s(i)), taking machine m(0) from its present state s(p) to the state

test s(i). The J-sequence must not contain any untested transitions which might be influenced by some of the faults f(1),...,f(NF).

4) Apply test t(i,j).

5) Find the propagating sequence D(i) which distinguishes s'(0), entered by m(0), from s'(i), entered by m(i). If no propagating sequence exists and the machines m(0) and m(i) are both strongly connected, the fault f(i) is undetectable. On the other hand, if either of them is not strongly connected, it is necessary to go back to step 1 and find a different test t(i,j). Only if all tests t(i,j) were exhausted and no state propagating sequence was found, can f(i) be said to be undetectable.

6) Choose the next fault for testing and return to 2.

An execution of the sequence on the tested machine should be stopped immediately if the observed output differs from the outputs of m(0). This indicates the presence of a fault and further results could be misleading. If only cells c(i) testing for individual state faults are desired, only steps 3 and 5 have to be executed for each fault f(i).

In the following example, a testing sequence for the given set of non-overlapping faults will be generated.

Example 3.3: Find a testing sequence for the machine given in Fig. 3.3, and the following set of faults:

Fault #	Location	Type
1	1	s-0
2	2	s-1
3	3	s-1
4	4	s-1
5	5	s-0
6	6	s-0

The state tables of the machines m(0) through m(6) are given in the following table.

Table 3.3 State tables of machines m(0),...,m(6)

	m(0)		m(1)		m(2)		m(3)		m(4)		m(5)		m(6)	
	0	1	0	1	0	1	0	1	0	1	0	1	0	1
A	A,0	B,0	A,0	B,0	A,0	B,1	A,0	B,0	B,0	B,0	A,0	A,0	A,0	B,0
B	A,0	A,1	A,0	B,1	A,0	A,1	A,1	A,1	A,1	A,1	A,0	A,1	A,0	A,0

The tests for the given faults as found in Example 3.1, are:

t(2): y,x=0,1

t(3): y,x=1,0

t(6): y,x=1,1

Faults f(1) and f(5) are state faults, having tests:

t(1): y,x=1,1

t(5): y,x=0,1

The fault f(4) can behave either as a state or as an output fault, depending on the test used. The tests are:

t(4,1): y,x=0,0

t(4,2): y,x=1,0.

Assuming that the machine is initially in state A, consider:

1) y=0, since the machine is in state A, therefore f(4) can be tested by applying x=0.

2) Apply x=1 which is a distinguishing sequence for the machines m(0) and m(4). If the output is correct, it means that the A,0 transition, executed in step 1, is correct; however the transition A,1 may be incorrect.

3) The second input, besides being a member of the distinguishing sequence, is also a test for faults 2 and 5 (y,x=0,1). Therefore if x=1 is applied, which is a distinguishing sequence for m(2) and m(5), and the output is correct, the A,1 transition must be correct too. The transition B,1 may be incorrect.

4) By the same reasoning, if x=1 is applied again, the B,1 transition is correct. It was already established that the A,1 transition is correct, if the previous response was correct.

5) It only remains to test for the output fault f(3) which can be done by setting x to 0. Because f(3) is an output fault, no other inputs are necessary.

The resulting testing sequence is:

x :	0	1	1	1	0
s of m(0):	A	A	B	A	B
z of m(0):	0	0	1	0	0
m(1):	0	0	1	1	
m(2):	0	1			
m(3):	0	0	1	0	1
m(4):	0	1			
m(5):	0	0	0		
m(6):	0	0	0		

Procedure 3.2 is relatively simple and easy to use. However, it will not work for some machines, because there may not be any transition which would satisfy all of the previously stated restrictions at the current point of the experiment. A general procedure can be deduced by observing the testing tree, shown in Fig. 3.4. This tree was obtained as follows: First, it is assumed that the machine is initially in the state denoted by S(i). If T(i,j) is applied and if f(i) is present, the machine enters the state s'(i) and starts a new branch b(i) in the testing tree. Otherwise the machine enters the state S'(0) in the branch b(0) corresponding to the machine m(0). Since the

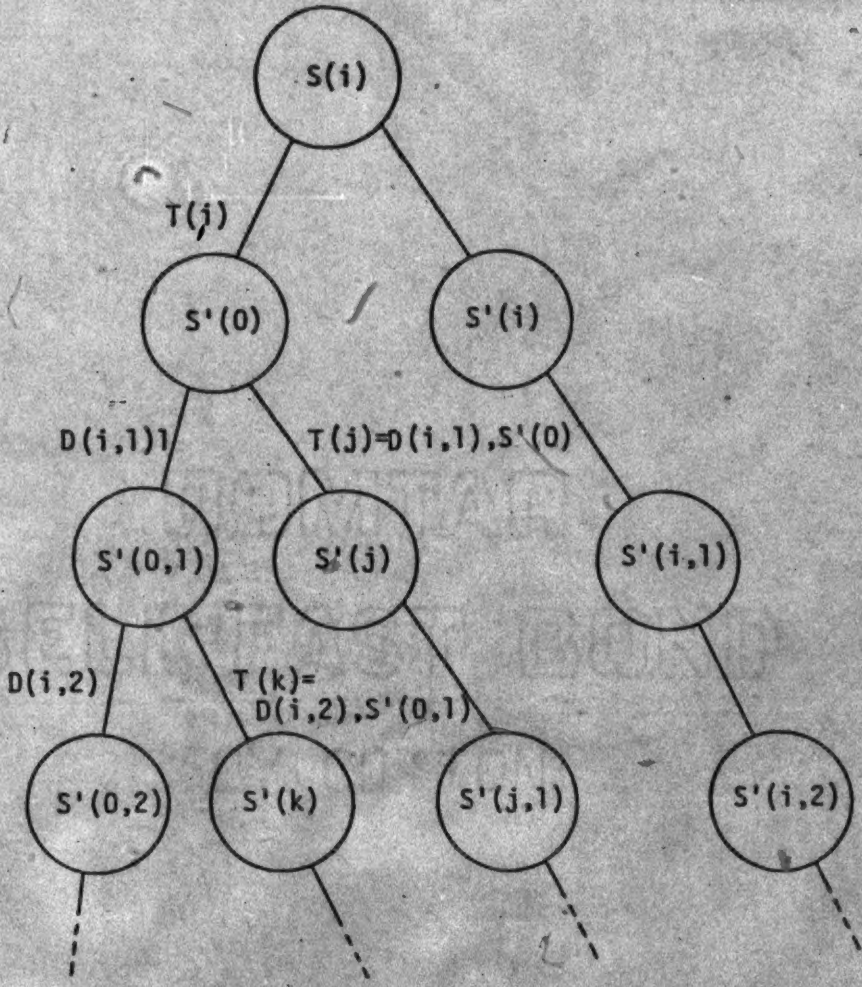

S(i)
T(j)
S'(0)
S'(i)
D(i,1)1
T(j)=D(i,1),S'(0)
S'(0,1)
S'(j)
S'(i,1)
D(i,2)
T(k)=
D(i,2),S'(0,1)
S'(0,2)
S'(k)
S'(j,1)
S'(i,2)

Fig. 3.4 A testing tree

set of faults is assumed to be non-overlapping, at most, one new branch corresponding to the state fault, can be initiated by each input in the leftmost branch b(0). The propagating sequence D(i) must now be applied in order to find out which branch of the tree the machine is actually following. Transitions in branches b(1),...,b(NF), resulting from the inputs of the propagating sequence, fall into three categories:

1) The transition is not influenced by any fault in F.

2) The transition was already completely tested.

3) The transition is in a branch indicating the presence of a certain non-overlapping fault and therefore no other faults may be present. The transition is then influenced by the fault which is currently tested.

In all three of the above cases the next states are uniquely determined, which means that all states of the b(1),...,b(NF) branches have only a single possible successor. Therefore no futher branching can occur, exept in the leftmost branch b(0), and then only if the transition does not fall under 1 or 2.

In the following example the testing tree, for the machine analyzed in Example 3.3, will be constructed.

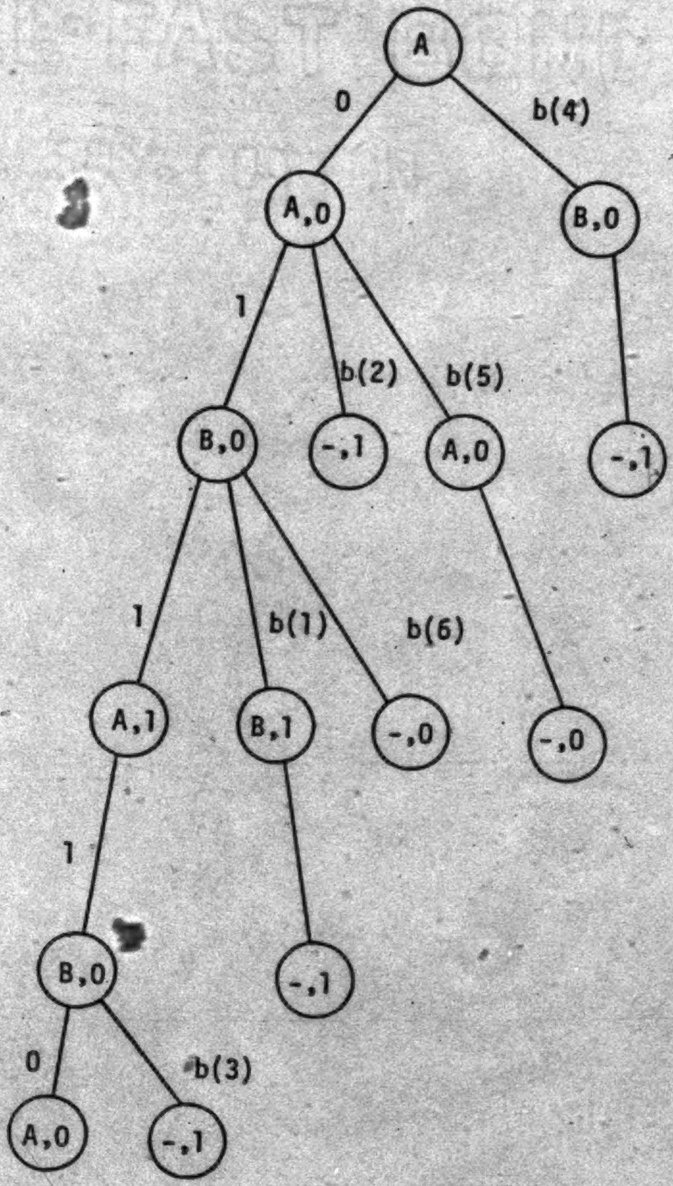

A
0
b(4)
A,0
B,0
1
b(2)
b(5)
B,0
-,1
A,0
-,1
1
b(1)
b(6)
A,1
B,1
-,0
-,0
1
B,0
-,1
0
b(3)
A,0
-,1

Figure 3.5 Testing tree for machine M3.2

Example 3.4: Construct the testing tree for the machine and set of faults given in the Example 3.3.

The resulting testing tree is shown in the Fig. 3.5. The distinguishing input 1 is used for propagating purposes. The order in which the individual tests are applied was governed primarily by convenience.

In section 4.2 state propagating sequences will be discussed in detail. From their definition it follows that any unused transition is a potentially useful member of the propagating sequence D(i). These transitions, however, must not result in the same state in b(i), the branch coresponding to the faulty machine m(i) and in b(0), the branch coresponding to good machine m(0), during the experiment. If branches b(0) and b(i) would enter the same state with the same output, i.e., the branches b(0) and b(i) would merge, then the input cannot be a member of the propagating sequence at the current point. The propagating sequences can be obtained by trial and error. A more systematic approach is described in 4.2. If the machine posseses a distinguishing sequence, its inputs are usually good choices for a propagating sequence.

The above discussion can be summarized in the following procedure.

Procedure 3.3:

1) Find all tests t(i,j) for every fault f(i) in the given set of faults F. Denote the tests obtained by T.

2) Select any state s(i), belonging to any test t(i,j) in T, which can be reached from the present state without going through any other test also contained in T.

3) Force the machine into state s(i).

4) Apply the input test U(i,j). This will start the new branch b(i) in the testing tree. Remove all tests t(i,1),...,t(i,NT(i)) from T.

5) Find the propagating sequence D(i) which distinguishes between states s'(0) and s'(i). If no propagating sequence exists, and if the machines m(0) and m(i) are both strongly connected, the fault f(i) is undetectable. If on the other hand either m(0) or m(i) are not strongly connected, one has to go back to step 1 and find a different test t(i). Only if all tests t(i,j) are exhausted and no state propagating sequence was found, can f(i) be pronounced as undetectable. After the application of D(i), the branch b(i) is terminated.

6) If all faults f(i), i=1,...,NF have been exhausted, stop. Otherwise return to 2.

The length of the sequence can be optimized if in step 2 one selects i pointing to the testing state which can be reached from the present state via the shortest possible transfer sequence.

As an alternative, it is possible to start with the application of the tests generated in step 1, until all of them are used. Naturally merging of tree branches may occur and it is necessary to keep track of branches which merged in this stage of the experiment. Tests that have initiated the branches which merged must be repeated and followed by the proper state propagating sequence. Only after all tests have been applied, is the propagating sequence used. The following procedure is based on this alternate approach.

Procedure 3.4:

1) Find at least one test t(i,j) for each fault in the given set of faults.

2) Select any i between 1 and NF which leads to the nearest unused testing state s(i).

3) Force the machine into S(i).

4) Apply the testing input U(i,j).

5) Select the next i and unless all values between 1 and NF have been covered, go to 6. Otherwise go to 7.

6) Apply t(i,j). If the application of this test has caused the branch b(0) to merge with any existing branches, note branch numbers of all branches which merged. If any one of the existing branches was distinguished from b(0), the branch is terminated. Return to 2.

7) Find the state propagating sequence for the branch with the smallest i of all branches which are still not identified.

8) Apply D(i). Again note all branches which merge with b(0). Upon the application of D(i) the branch b(i) is terminated. Go back to 7 until all branches have been terminated.

9) Use procedure 3.3 for all faults corresponding to branches which merged.

The following example demonstrates the application of Procedures 3.3 and 3.4.

Example 3.5: Given the machine pictured in the Fig. 3.6, find a testing sequence for the following set of faults:

Fault #	Location	Type
1	1	s-0
2	3	s-1
3	18	s-0

The transitions influenced by the above faults are: Under fault f(1), transition from D under input 10 results in state D with output 0. This can be written as

$D,10 \rightarrow D,0$

Similarly:

$D,11 \rightarrow D,1.$

From this it follows that the tests are:

$t(1,1): y(1),y(2),x(1),x(2)=1,0,1,0$

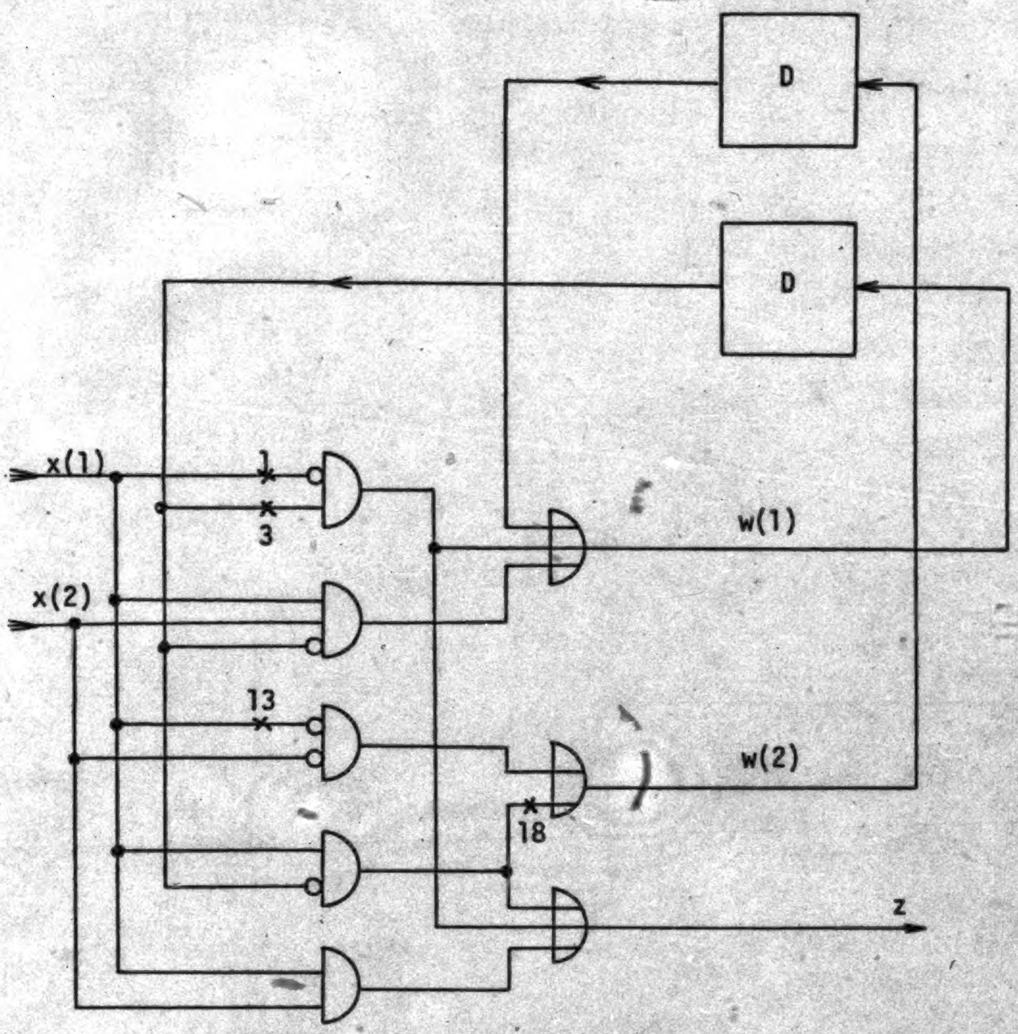

D
D
x(1)
x(2)
1
3
13
18
w(1)
w(2)
z

Fig. 3.6 Circuit diagram for machine M3.3

t(1,2): y(1),y(2),x(1),x(1)=1,0,1,1

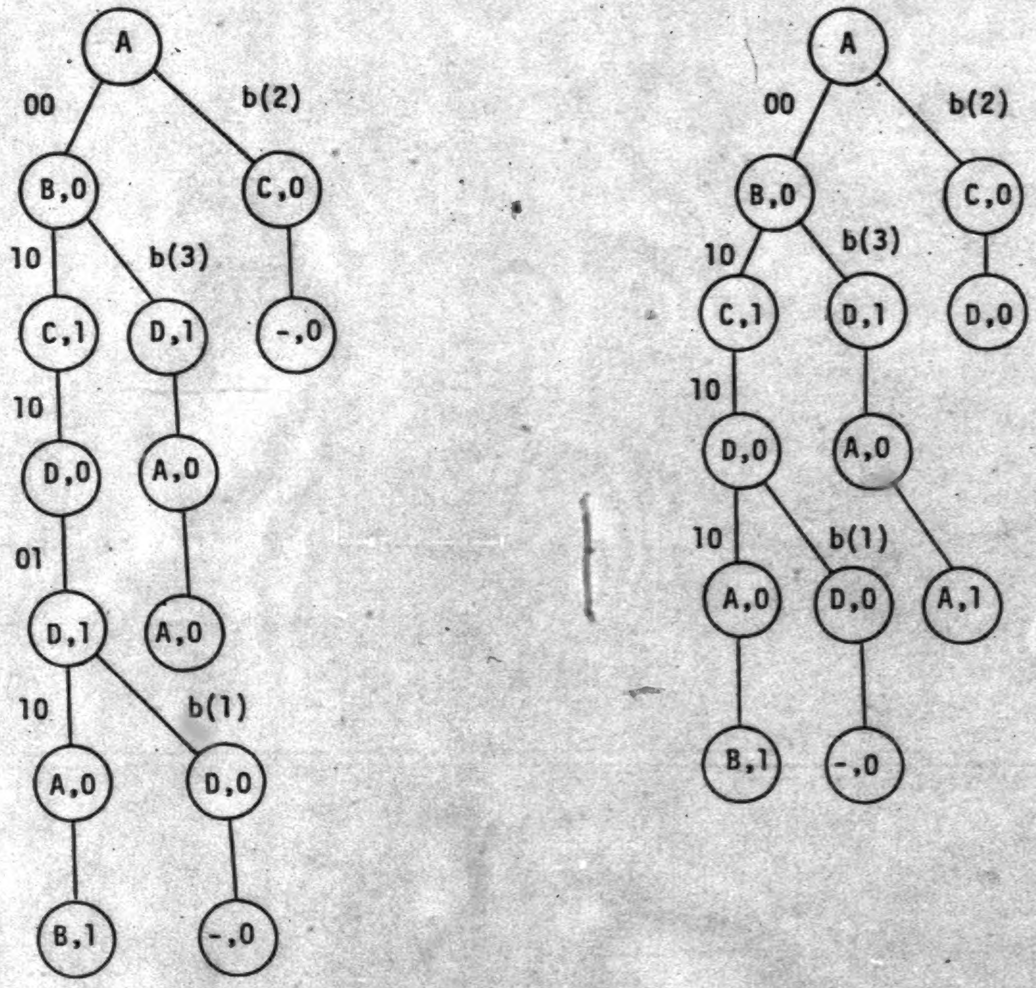
Similarly:

Under fault f(2): A,00>C,0, A,01>D,0

Under fault f(3): A,10>A,1, A,11>D,1, B,10>D,1, B,11>D,1

The set of faults is therefore non-overlapping. The machine has two distinguishing sequences, (2,1) and (2,2) or in binary (10,01) and (10,10). Using these sequences for branch identification in procedure 3.3, one obtains the testing tree shown in Fig. 3.7a. This tree was obtained under the assumption that the machine is initially in state A. Then the cells, each consisting of the test for one fault followed by one of the distinguishing sequences, were applied. State A is a state test for f(2) or f(3). The test for f(2) was arbitrarily applied first. This test was followed by the distinguishing sequence in order to propagate the outcome of the test to the output. The first input of the propagating sequence not only distinguishes between branch b(0) and b(2) but also forms together with state B a test for f(3). The next two inputs propagate the outcome of the test for f(3) to the output. Finally, the cell testing for f(1) is applied. The resulting testing sequence is (0,2,2,1,2,2), which expressed in binary is (00,10,10,01,10,10).

The testing tree resulting from procedure 3.4 appears in Fig. 3.7b. This testing tree was again obtained under the assumption that the machine is initially in the state A. Then


A
00
b(2)
B,0
C,0
10
b(3)
C,1
D,1
-,0
10
D,0
A,0
01
D,1
A,0
10
b(1)
A,0
D,0
B,1
-,0
A
00
b(2)
B,0
C,0
10
b(3)
C,1
D,1
D,0
10
D,0
A,0
10
b(1)
A,0
D,0
A,1
B,1
-,0

a) b)

Fig. 3.7 Testing trees for machine M3.3

the test for f(2) was immediately followed by the test for f(3) without any intention of applying any propagating sequence at this point. This test was again immediately followed by the transfer sequence into the state test for f(1), and the test for f(1). The last input propagates the outcome of the last test to the output. The resulting testing sequence is (0,2,2,2,2)=(00,10,10,10,10).

Testing procedures for machines with overlapping state faults will be discussed in the next section.

3.4 Testing sequences for machines with overlapping state faults.

The problem of testing machines with overlapping state faults is very similar to the testing of machines when only non-overlapping faults are considered. The only difference results from the fact that in this case there can be several different responses resulting from a single test, depending on the fault present. Therefore, considering the testing tree described in the previous section, there can be more than one branch initiated by each test in the leftmost branch b(0) of the tree, (see Fig. 3.8). The testing sequences can be developed according to procedures 3.3 or 3.4, provided that an attempt is made to follow every unidentified branch of the tree by the appropriate state propagating sequence.

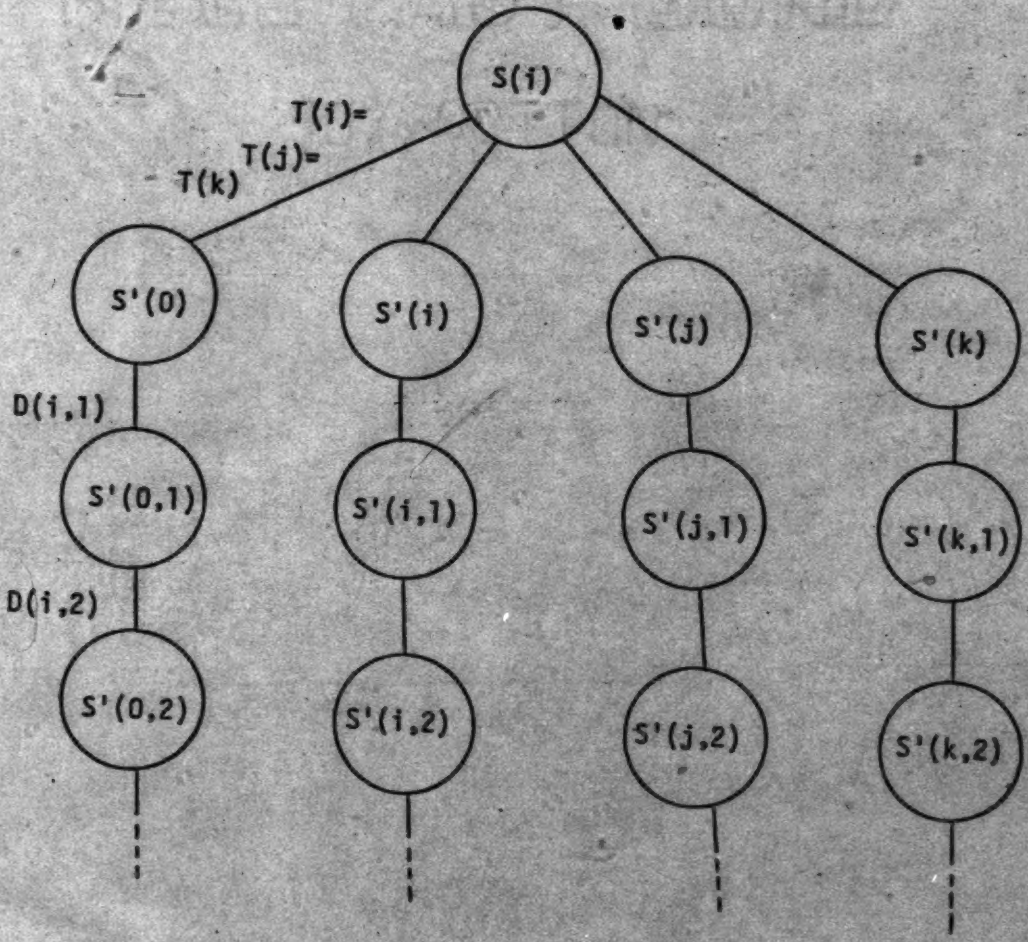

S(i)
T(i)=
T(j)=
T(k)
S'(0)
S'(i)
S'(j)
S'(k)
D(i,1)
S'(0,1)
S'(i,1)
S'(j,1)
S'(k,1)
D(i,2)
S'(0,2)
S'(i,2)
S'(j,2)
S'(k,2)

Fig. 3.8 A testing tree for machines with overlapping state faults

It should be noted that by treating branches b(1),...,b(NF) in the same way as b(0) branch, namely by examining whether any transition in these branches constitutes test or not, procedures 3.3 and 3.4 can be used to generate testing sequences for intermittent faults.

The following example will illustrate the generation of the testing sequence for the machine with overlapping set of faults.

__Example 3.6__: Find a testing sequence for the machine given in the Fig. 3.5, assuming the following set of faults:

Fault#	Location	Type
1	1	s-0
2	3	s-1
3	18	s-0
4	13	s-1

Transitions influenced by faults f(1),...,f(3) and the tests were given in Example 3.5. Under f(4) A,C0>A,0, B,00>D,0, C,00>D,1, D,00>D,1 and the tests are:

t(4,1): 0,0,0,0

t(4,2): 0,1,0,0

t(4,3): 1,1,0,0

t(4,4): 1,0,0,0

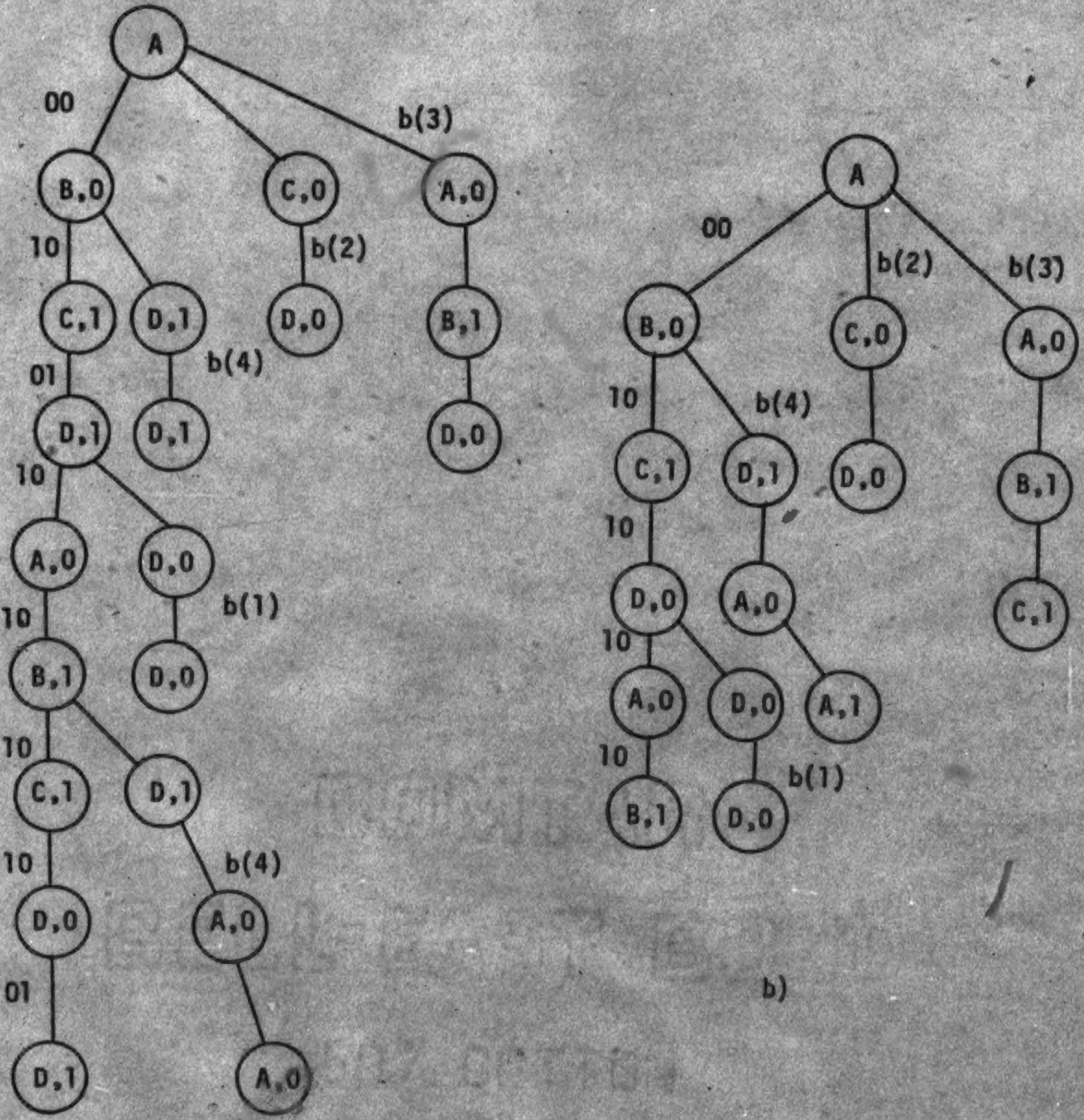
The set of faults is overlapping (faults 2 and 4).

3

Using Procedure 3.3, the testing tree shown in Fig. 3.9a is obtained. The first transition tests for f(2) and f(4). It is followed by the first member of the propagating sequence D(2) and D(4). The transition triggered by this input also tests for f(3). The third input is the second member of the propagating sequence D(4). This input, however, merges the branch b(4) with the branch b(0), which means that the test for f(4) will have to be repeated. The forth input tests for f(1). It is followed by the first member of the propagating sequence D(1). The sixth input tests for f(4). The last two inputs constitute the propagating sequence D(4).

On the other hand, using procedure 3.4 the testing tree shown in Fig. 3.9b is obtained. The first transition tests for f(2) and f(4). The second transition tests for f(3). The next input constitutes the transfer sequence to the nearest unused testing transition. The forth transition tests for f(1) and finally the last input is the first member of D(1).

The testing trees shown in Fig. 3.9 are not the only possible testing trees for the given machine and set of faults. Procedures 3.3 and 3.4 allow an arbitrary choice of inputs in certain stages of the experiment, as demostrated in Example 3.5.


A
00
b(3)
B,0
C,0
A,0
10
b(2)
C,1
D,1
D,0
B,1
b(4)
01
D,1
D,1
D,0
10
A,0
D,0
b(1)
10
B,1
D,0
10
C,1
D,1
10
b(4)
D,0
A,0
01
D,1
A,0
A
00
b(2)
b(3)
B,0
C,0
A,0
10
b(4)
C,1
D,1
D,0
B,1
10
D,0
A,0
C,1
10
A,0
D,0
A,1
10
b(1)
B,1
D,0

a)

b)

Fig. 3.9 Testing trees for machine M3.3

There is another way in which testing sequences for overlapping set of faults can be developed. A given set of faults F can be divided into subsets F(1),...,F(NO), each subset being non-overlapping. Then, assuming that the machine can be reinitialized by the procedure R, and that E(i) is the testing sequence for F(i), the machine can be tested by the sequence

R, E(1), R, E(2),...,R, E(NO-1), R, E(NO).

The generation of testing sequences as outlined in Procedures 3.3 and 3.4 will be further discussed in detail in the following chapter.

Chapter 4

GENERATION OF TESTING SEQUENCES

In this chapter a description of a practical use of the testing methods developed in the previous chapter is given. Discussed in detail are the problems of selection of necessary tests from a given complete set of tests and that of obtaining responses to transitions in faulty machines. Also a procedure which generates state propagating sequences will be given.

The algorithms for the generation of testing sequences for sequential machines, require the following tasks to be accomplished:

1) Generate a complete set of tests for the combinational part of the sequential circuit.

2) Determine which transitions of the tested machine are influenced by the given set of faults.

3) Determine which tests will result in output faults and which will result in state faults.

4) Establish how states will be changed by state faults.

5) Generate state propagating sequences as required.

As stated in Chapter 3, numerous methods for the generation of tests of combinational circuits are available [1,2,3,4,5,7,8,9,10,15,17,19,21], and will not be discussed here.

The tasks mentioned under 2,3 and 4 above, yield information which is necessary to construct the branches of a testing tree. They identify a fault set in terms of the transitions changed by the set and therefore perform what will be called fault set identification.

4.1 Fault set identification

The first task of fault set identification is straightforward. Every test consists of two parts, an input test and a state test (see Chapter 3). The state test is applied to the feedback inputs of the tested circuit, while the input test is applied to the external inputs. Therefore the state test S(i) and the input test U(i) correspond to the state s(i) and the input u(i) of the transition, influenced by the fault f(i).

The simplest method for determining which fault will result in an output fault and which in a state fault is to simulate the behaviour of the tested machine for each fault. This yields the state tables of machines m(0) through m(NF). Then if under the fault f(i), an output of a certain transition is different from an output of the same transition in $m(0)$, the corresponding test will result in the output fault. Analogically, the test $t(i,j)$, which will result only in a change of state, will test for a state fault. The outcome of every transition under any fault would also be known from the simulation. The simulation, however, requires a large amount of computation. A better approach is given in the following paragraph.

Assume that the test $t(i,j)$ for the fault $f(i)$ is applied to the tested circuit. Then it can be determined, if $t(i,j)$ sensitizes the path:

1) between $f(i)$ and an output of the circuit, or
2) between $f(i)$ and a feedback output of the circuit, or
3) between $f(i)$ and both an output and a feedback output.

In the first case, the fault will obviously behave as an output fault, in the second case it will be a state fault, and finally in the third case it can be treated as either a state or an output fault. In the last case, however, it is always advantageous to treat the fault as an output fault because then there is no problem associated with propagating the resulting state to an output. The results of this process, which is only performed for tests actually used in the experiment, can be tabulated in a form of a circuit state and output fault dictionary. The table will have NF rows, one for each fault, and two columns, one for each fault type.

The first column will contain those tests under which the fault will behave as an output fault, while the second one will contain those tests under which the fault will behave as a state fault. A test will be listed in both columns if it results in state and output faults. Such a dictionary will be similar to the ones in Table 4.2.

To find the successors of each faulty transition, the above scheme can be carried somewhat further. Assume that the test $t(i,j)$ for the state fault $f(i)$, is applied to the circuit, and if the machine operates correctly, the corresponding transition yields a state $s(i)$ with a state assignment of:

$$S(i)=\{s(i,1),s(i,2),\ldots,s(i,NB)\}.$$

The test $t(i,j)$ sensitizes the paths between the site of the fault $f(i)$ and at least one of the feedback outputs $w(1),\ldots,w(NB)$. The state $s'(i)$, which will result from $t(i,j)$ if $f(i)$ is present, will have all the values $s'(i,k)=s(i,k)$ for all feedback outputs $w(i,k)$ to which there are no sensitized paths leading from the location of the fault. On the other hand, the feedback outputs $w(i,l)$, to which there is a path sensitized by the test $t(i,j)$, will have values $s'(i,l)=\overline{s(i,l)}$ if the fault is present. The paths which are sensitized by the individual tests, can be tabulated in the *path and transition fault dictionary*, similar to the one in Table 4.3 and Table 4.4. Such a dictionary is then used for determining the resulting state under any fault in a given set. In this approach, it is advantageous that only the transitions resulting from the tests, which are actually used in the experiment, must be simulated. Also only the state table of m(0) and the circuit fault dictionary must be kept instead of state tables for all machines m(0),...,m(NF).

Example 4.1: In this example, the problem is to find a testing experiment for all stuck type faults on gates 1,...,5 of the machine given in Fig. 4.1 and Table 4.1. The tests can be obtained using any method for the generation of tests in combinational circuits*.

All possible stuck type faults can be covered by considering s-1 faults on all non-inverting input leads, s-0 faults on all inverted input leads, and s-0 faults on all output leads. For example f(1) is an s-0 fault, f(2) is an s-1 fault and f(3) is an s-0 fault. Faults f(4) through f(10) are state faults because the only path(s) which can be sensitized from the locations of these faults lead to the feedback outputs of the circuit. Faults f(14), f(15) and f(16) will behave as output faults because all possible paths from these faults lead to the

---

* The tests will be expressed according to the following formula:

test=x(1)*8+x(2)*4+y(1)*2+y(2),

where: x(1),...,y(2) are binary inputs and binary feedback inputs and '*' indicates multiplication.

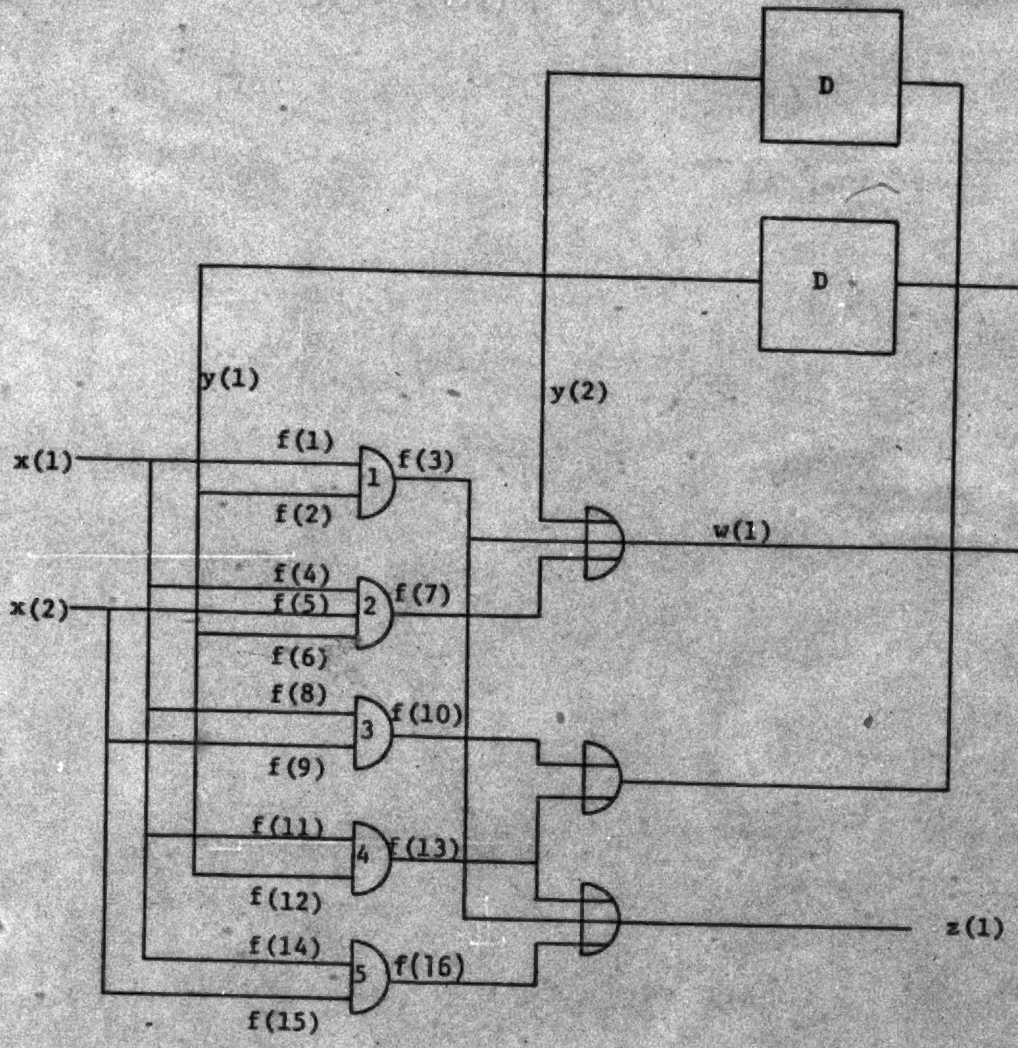

D
D
y(1)
y(2)
x(1)
f(1)
1
f(3)
f(2)
w(1)
f(4)
x(2)
f(5)
2
f(7)
f(6)
f(8)
3
f(10)
f(9)
f(11)
4
f(13)
f(12)
z(1)
f(14)
5
f(16)
f(15)

Fig. 4.1 Circuit diagram of machine M4.1

Table 4.1 State table of machine M4.1

S/I	0	1	3	2
00=A	B,0	A,0	C,1	B,1
01=B	C,0	D,0	C,1	C,1
11=C	C,1	D,1	D,1	D,0
10=D	C,1	D,1	A,1	A,0

outputs of the circuit. Finally, faults f(1), f(2), f(3), f(11), f(12) and f(13) can behave as either state faults or as output faults, depending on the test used. This occurs because some tests sensitize paths between the fault and the feedback variables, while others sensitize the paths between the fault and outputs. Faults and tests for this example are summarized in Table 4.2.

The essential tests are 4,8,14 and 12. They cover all faults except f(3),f(8),f(10) and f(15). They must be supplemented by at least one test for each of these faults. The selection of these tests, however, is best left until all essential tests are applied. In this process some other tests are inadvertedly applied as well, and therefore some of the above faults may be detected by the time all essential tests are applied.

Now an experiment can be developed by using Procedure 3.3

Table 4.2 Test table for machine M4.1

Fault#	Gate	Test	Transition	Type
1	1	14	D,11	S
		10	D,10	O
		11	C,10	O
2		0	A,00	O
		1	B,00	O
		1	B,00	O
		5	B,01	O
3		2	D,00	O
		3	C,00	O
		5	B,01	O
		6	D,01	O
4	2	4	A,01	S
5		8	A,10	S
6		14	D,11	S
7		12	A,11	S
8	3	10	D,10	S
		11	C,10	S
9		4	A,01	S
		5	B,01	S
		6	D,01	S
		7	C,01	S
10		4	A,01	S
		1	B,00	S
		2	D,00	S
		3	C,00	S
11	4	0	A,00	O
		1	B,00	O
		4	A,01	O
		4	A,01	O
12		10	D,10	O
		11	C,10	O
		14	D,11	O
12	4	15	C,11	S
13		8	A,10	O
		9	B,10	O
		12	A,11	O
		13	B,11	O
14	5	4	A,01	O
		5	B,01	O
15		10	D,10	O
		11	C,10	O
16		14	D,11	O
		15	C,11	O

or 3.4. Two experiments are shown in Figs.4.2 and 4.3. In Fig. 4.2 Procedure 3.3 was used, while Fig. 4.3 was obtained from Procedure 3.4. Any transition which appears in these testing sequences must fulfill one or more of the following purposes: it is a transition tests for a fault, it propagates a faulty state to the output, or it transfers the machine into next state test. The tree transitions denoted with the test number followed by the corresponding transition in the brackets, for example 14(D,11), were selected for testing purposes. The transitions, denoted with the transition code followed by a corresponding test number, as in A,10(8), were selected for the purpose of branch(state) distinguishing. Some of them may correspond to the test for a fault which has not been yet tested, in which case their dual purpose was taken into consideration. Namely a new branch is started for each fault which is also tested by such a transition. The transitions denoted simply with the transition code, for example B,01, are transfer transitions, and do not correspond to any test for any fault in the given set of faults.

The successor states of the transitions, executed in the branch b(0) of the tree, are obtained directly from the state table of m(0). The successors in other branches are obtained by taking the bit pattern of the state, which would result in m(0), and inverting the bit corresponding to the sensitized path.

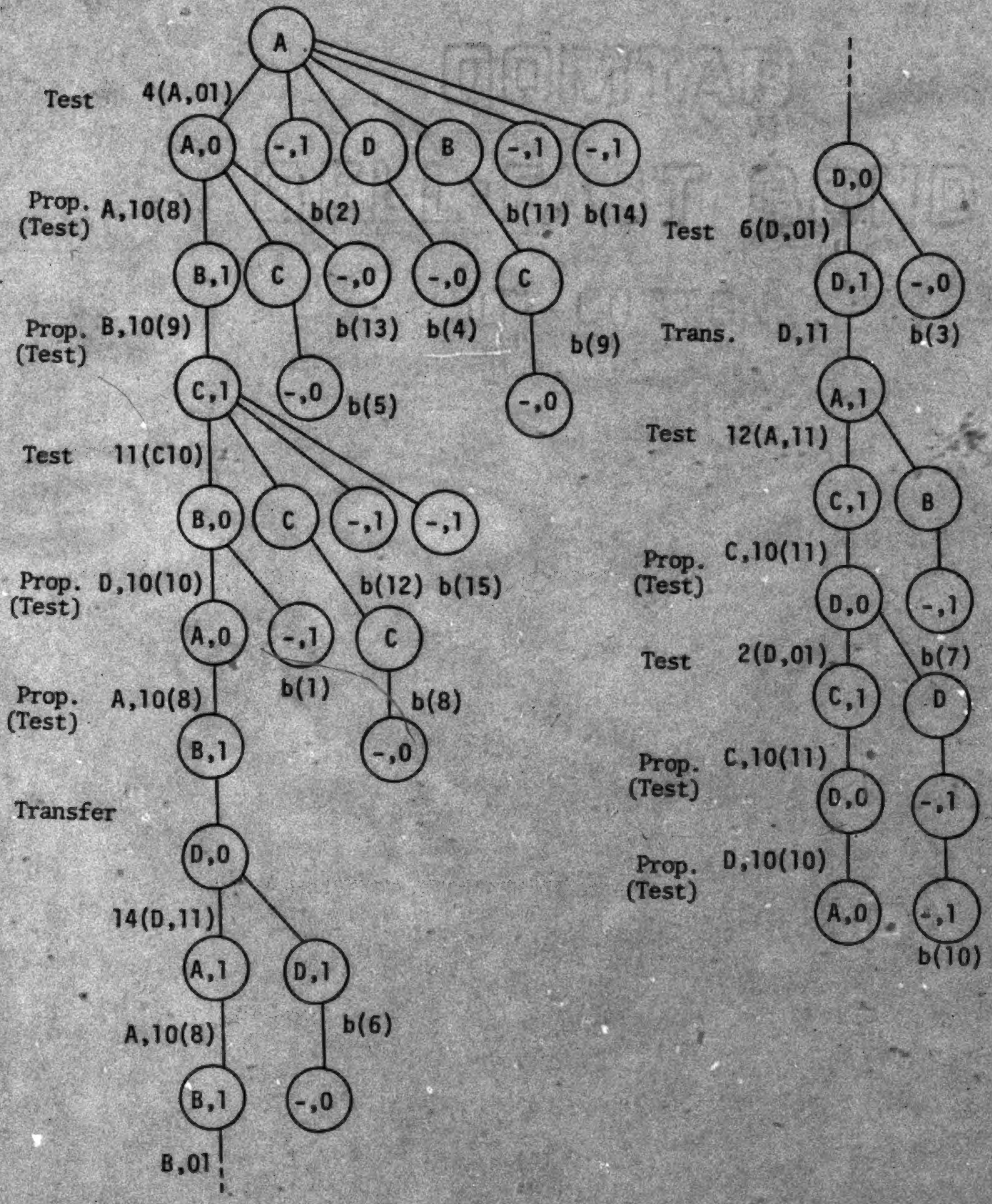

A
Test 4(A,01)
A,0
-,1
D
B
-,1
-,1
Prop. (Test) A,10(8)
b(2)
b(11) b(14)
B,1
C
-,0
-,0
C
Prop. (Test) B,10(9)
b(13) b(4)
b(9)
C,1
-,0
b(5)
-,0
Test 11(C10)
B,0
C
-,1
-,1
Prop. (Test) D,10(10)
b(12) b(15)
A,0
-,1
C
Prop. (Test) A,10(8)
b(1)
b(8)
B,1
-,0
Transfer
D,0
14(D,11)
A,1
D,1
A,10(8)
b(6)
B,1
-,0
B,01
D,0
Test 6(D,01)
D,1
-,0
Trans. D,11
b(3)
A,1
Test 12(A,11)
C,1
B
Prop. (Test) C,10(11)
D,0
-,1
Test 2(D,01)
b(7)
C,1
D
Prop. (Test) C,10(11)
D,0
-,1
Prop. (Test) D,10(10)
A,0
-,1
b(10)

Fig. 4.2 A testing tree for machine M4.1

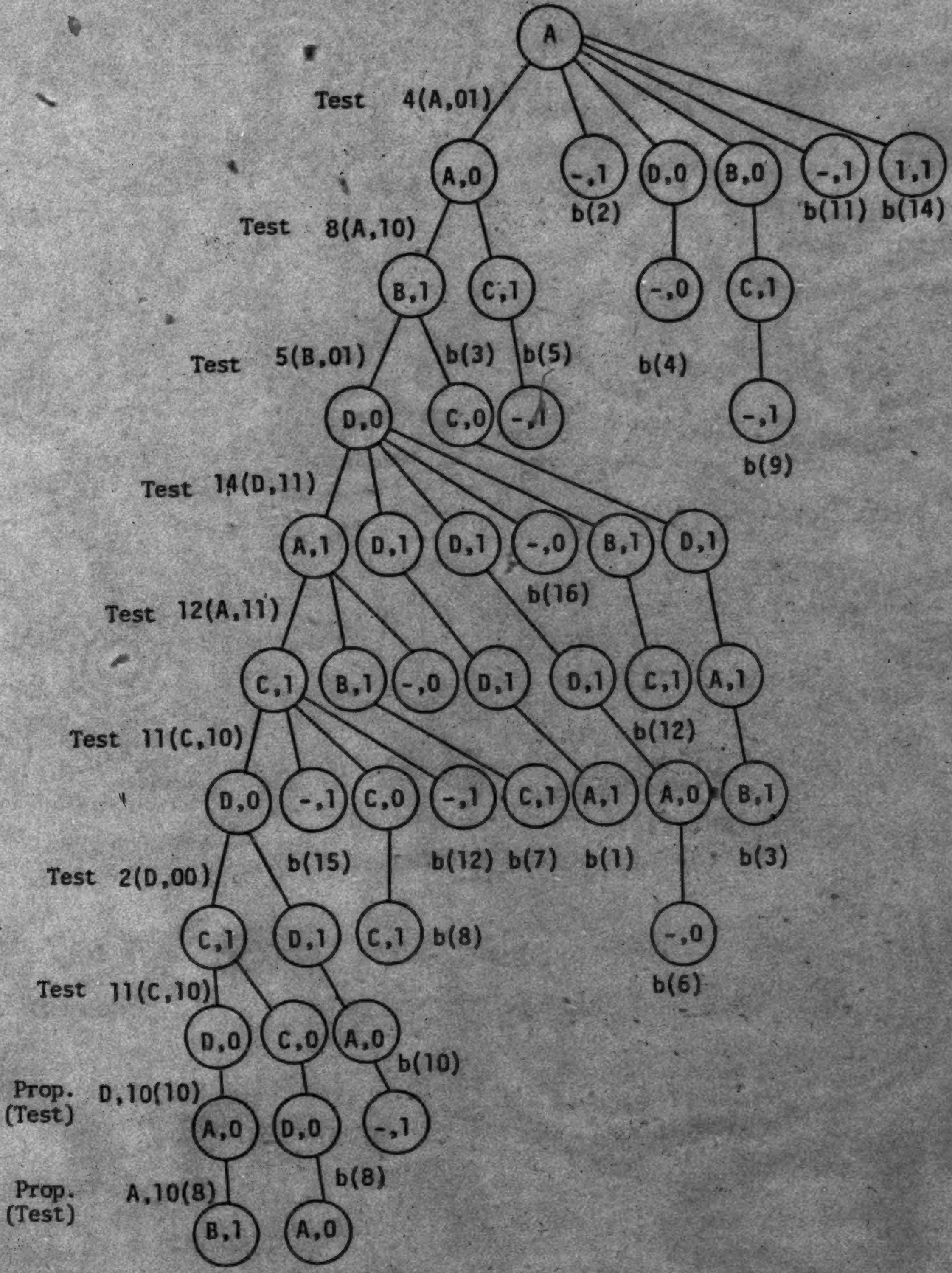

A
Test 4(A,01)
A,0
-,1
D,0
B,0
-,1
1,1
b(2)
b(11)
b(14)
Test 8(A,10)
B,1
C,1
-,0
C,1
Test 5(B,01)
b(3)
b(5)
b(4)
D,0
C,0
-,1
-,1
b(9)
Test 14(D,11)
A,1
D,1
D,1
-,0
B,1
D,1
b(16)
Test 12(A,11)
C,1
B,1
-,0
D,1
D,1
C,1
A,1
b(12)
Test 11(C,10)
D,0
-,1
C,0
-,1
C,1
A,1
A,0
B,1
b(15)
b(12)
b(7)
b(1)
b(3)
Test 2(D,00)
C,1
D,1
C,1
b(8)
-,0
b(6)
Test 11(C,10)
D,0
C,0
A,0
b(10)
Prop.
(Test)
D,10(10)
A,0
D,0
-,1
b(8)
Prop.
(Test)
A,10(8)
B,1
A,0

Fig. 4.3 A testing tree for machine M4.1

Each path leads to one feedback output, which represents one bit in the state pattern. The paths sensitized by each state test used in the testing sequence, developed according to Procedures 3.3 and 3.4, and the resulting transitions are shown in Table 4.3 and Table 4.4, respectively. The transitions are listed in the order in which they occur in the experiment.

To complete the testing sequence, it is necessary to follow some tests by an appropriate state propagating sequence. In this example the state propagating sequences will be obtained by trial and error, (see Chapter 3, Sections 1 and 3). A distingushing sequence will be used as an initial guess for the propagating sequence in this example.

The testing trees shown in Fig. 4.2 and 4.3 are not the only possible testing trees which can be obtained from Procedures 3.3 and 3.4 for the given set of faults. At certain stages of the experiment the choice of inputs is arbitrary, resulting in many possible trees.

The testing sequence resulting from Procedure 3.3 is 01,10,10,11,10,10,01,11,10,01,01,11,11,11,01,11,10. A shorter sequence, obtained from Procedure 3.4, is 01,10,01,11,11, 10,00,10,10,10.

In the previous example and generally whenever Procedures 3.3 and 3.4 are used, it is necessary to generate propagating

Table 4.3 Path and transition fault dictionary for machine M4.1

Fault	Path	Transition effected
f(4)	w(1)	A,01->D
f(9)	w(2)	A,01->B
f(5)	w(1)	A,10->C
f(8)	w(2)	C,10->C
f(6)	w(1)	D,11->D
f(7)	w(1)	A,11->B
f(10)	w(2)	D,00->D

Table 4.4 Path and transition fault dictionary for machine M4.1

Fault	Path	Transition effected
f(4)	w(1)	A,01->D
f(9)	w(2)	A,01->B
f(5)	w(1)	A,10->C
f(3)	w(1)	B,01->C
f(1)	w(1)	D,11->D
f(6)	w(1)	D,11->D
f(12)	w(2)	D,11->B
f(7)	w(1)	A,11->B
f(8)	w(2)	C,10->C
f(10)	w(2)	D,00->D

sequences. In the following section a systematic approach to the generation of state proagating sequences will be described.

4.2 Generation of propagating sequences

State propagating sequences are needed in order to distinguish between branches b(0) and b(i) in the testing tree. Each branch b(i) is associated with some fault f(i), and therefore by distinguishing branch b(i) from branch b(0) it is possible to distinguish between the faulty machine m(i) and the good machine m(0).

As demonstrated by the previous example, special state differentiating sequences are often not needed to identify the particular branch of the testing tree. This occurs because the tests T, which are used to detect the faults, also have inherent distinguishing properties which can be utilized. Information deduced from responses to the set of tests is often sufficient to distuishing between the branches b(1),...,b(NF) of faulty machines m(1),...,m(NF) and the branch b(0) of the good machine. Propagating sequences are only necessary in cases where the distinguishing process was not completed after all necessary tests were applied. In such cases, the following procedures are likely to yield the most useful inputs for distinguishing purposes.

In the case of machines with distinguishing sequences (DS), the distinguishing sequence cannot be used instead of the propagating sequence indiscriminately because:

a) a distinguishing sequence of the faulty machine m(i) may be different from that of the good machine m(0),

b) the faulty machine may not have a distinguishing sequence at all.

The blind use of a DS could therefore result in serious errors since it could result in a merging (see Chapter 3) of branches of the testing tree.

To construct a propagating sequence D(i), i=1,...,NF, which distinguishes between branches b(i) and b(0), one has to find inputs which will result in output sequences which are different for the b(0) branch of the tree and any other branch b(i). Under no circumstances can the machine corresponding to the branch b(0) enter the same state as any machine represented by other branches b(1),...,b(NF).

If a good machine m(0) has a distinguishing sequence, then because the faulty machines m(i) will generally differ from the good machine m(0) in only a few transitions, the inputs of the DS should be considered as candidates for a propagating sequence. With this assumption in mind, one can propose the following procedure for the generation of a propagating sequence for any state s'(i).

Procedure 4.1:

1) Set j=1.

2) Select the j-th, initially j=1, input of the DS.

3) Examine this input for validity, i.e., it must not merge good and faulty branches.

4) If the input is valid, it becomes the k-th member, initially k=1, of the detecting sequence. Increment k.

Otherwise if the inputs of DS are still not exhausted increment j and go to 2, otherwise go to 5.

5) If a state s'(i) is still not identified, find an input which will take the machine to the state from which all possible inputs were not yet applied. If such an input is found, go to 2. If not, the result of the test cannot be propagated to the output and therefore the fault f(i) cannot be detected.

The advantages of this procedure are:

a) It is only necessary to remember the state table of the good machine m(0) and calculate the results of transitions sensitive to the given set of faults, for only those transitions actually used in the experiment.

b) It is only necessary to find the DS for the good machine m(0).

c) The sequence will tend to be of minimal length, provided that the above assumption is correct. This should be the case for a majority of faults.

In the case of machines without a DS, a partial distinguishing sequence (PDS) can be constructed in the same way as a DS. But in this case not all states can be uniquely identified by a PDS. The inputs of a PDS, however, can be used in the same way that the inputs of a DS were used in the procedure described above.

Chapter 5

FAULT DIAGNOSIS

In this chapter the problem of fault diagnosis will be discussed. The purpose of fault diagnosis is to identify the fault which causes a machine to malfunction. An input sequence that performs diagnosis is called a fault diagnosing sequence. Every input sequence which locates faults also distingishes between every faulty machine and the good machine, and therefore it must also detect all faults.

A machine can be tested in one of two ways:

1) A fault diagnosing sequence is used to perform both fault detection and fault location.

2) A fault detecting sequence is supplemented by a fault diagnosing sequence which locates faults which are not located during the execution of the fault detecting sequence.

The second approach is usually preferable for the following reasons. The diagnosing sequence for given faults tends to be longer than the fault detecting sequence for the same number of faults. If the fault detecting sequence does not indicate the presence of any fault, there is no need to execute the locating sequence.

In the following discussion, Procedures 3.3 and 3.4 will be extended to yield suplementary fault diagnosing sequences. The following modifications have to be made:

1) Test selection. For fault detection, the test which detects the most faults which have not yet been detected, is the best choice. On the other hand, for diagnosis the test which distinguishes the most faults which have not yet been distinguished, should be selected first. For example, assume that after the detection sequence has been executed, the faults are partitioned into

$$P=\{\overline{f(1), f(2)}; \overline{f(3), f(4), f(5)}; \overline{f(6), f(7)}; \overline{f(8)}\},$$

where the faults appearing in the same partition block respond to the detecting sequence with the same outputs. Then clearly any additional test which distinguishes between the faults in different partition blocks, say between f(1) and f(3), is of no value. As a result, the test which distinguishes between the greatest number of faults appearing in the same partition block is the optimal choice.

2) Propagating sequences. Since it is now necessary to distinguish between all branches of the testing tree, the propagating sequences must not merge any branches of the tree with other branches, not only the b(0) branch as was required for fault detection. In other words in order to distinguish

between faults f(i) and f(j), an additional restriction that branch b(i) corresponding to the fault f(i) must not be allowed to merge with branch b(j), must be added to Procedure 4.1. Therefore if it is required that a fault locating experiment distinguishes among all faults, i.e., if full diagnosis is necessary, then either no branches in the testing tree can merge, or the tests corresponding to any branches which merged must be repeated.

If only partial diagnosis is to be performed it is only necessary to distinguish between groups of faults, like faults associated with the same integrated circuit or with the same circuit board. Then it is only necessary to check for merging between the branches belonging to the different groups, since the branches belonging to the same group can merge.

The following example will illustrate a problem of supplementing the fault detecting sequence with additional inputs which provide the required diagnostic information.

Example 5.1: In this example, supplement the fault detecting sequence obtained in the Example 4.1, to perform the following additional tasks:

a) Distinguish between faults f(2), f(11), plus f(14), and

b) Distinguish between faults f(1), f(3), f(7), f(12) and f(15).

Part a) can be solved as follows: From the tree in the Fig.4.5, obtained in Example 4.1 by using Procedure 3.3, it follows that after execution of the detecting experiment the faults are partitioned into the following blocks:

b(1)= f(2), f(11), f(14)
b(2)= f(13), f(4)
b(3)= f(5), f(9)
b(4)= f(12), f(15)
b(5)= f(1)
b(6)= f(8)
b(7)= f(16)
b(9)= f(6)
b(10)= f(3)
b(11)= f(7)
b(12)= f(10)

Faults f(2), f(11) and f(14) are all in the same partition block and therefore additional tests will be required in order to disitinguish between them. The tests for these faults, as established in Example 4.1, are:

f(2): 0, 4
f(11): 4, 5
f(14): 4, 5

From the above, and from the fact that f(11) and f(14) are output faults resulting in the same output, it follows that it is impossible to distinguish between them. To distinguish between f(2) and the other two faults, it is necessary to use test 0. Assuming that the machine can be reinitialized to state A, this input will result in the situation pictured in Fig. 5.1, which is sufficient to accomplish the desired task.

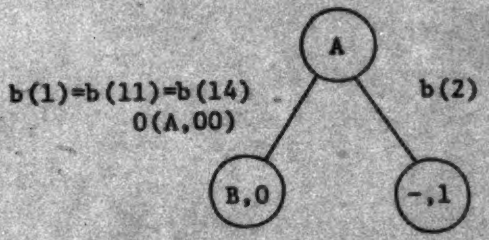


Fig. 5.1 Testing tree for Example 5.1a

Now consider the second part of this example. From the testing tree obtained from Procedure 3.4 and pictured in Fig.4.6, it follows that the faults are partitioned by the detecting sequence into the following blocks:

b(1)= f(2), f(11), f(14)
b(2)= f(4)
b(3)= f(9), f(5)

b(4)= f(16)

b(5)= (13)

b(6)= f(15), f(12), f(7), f(3), f(1)

b(7)= f(6)

b(8)= f(10)

b(9)= f(8)

Again all of the faults to be distinguished are found in the same partition. The tests for these above faults are:

f(15): 10, 11

f(12): 10, 11, 14, 15

f(7): 12

f(3): 2, 6

f(1): 10, 14

The application of test 11 will lead to the following partition blocks:

f(15), f(12)

f(7), f(3), f(1)

Then test 14 will result in:

f(15)

f(12)

f(1)

f(7), f(3)

Finally test 12 will distinguish between f(7) and f(3). Assuming that the machine can be reinitialized to the state A, the machine will behave as pictured in Fig. 5.2.

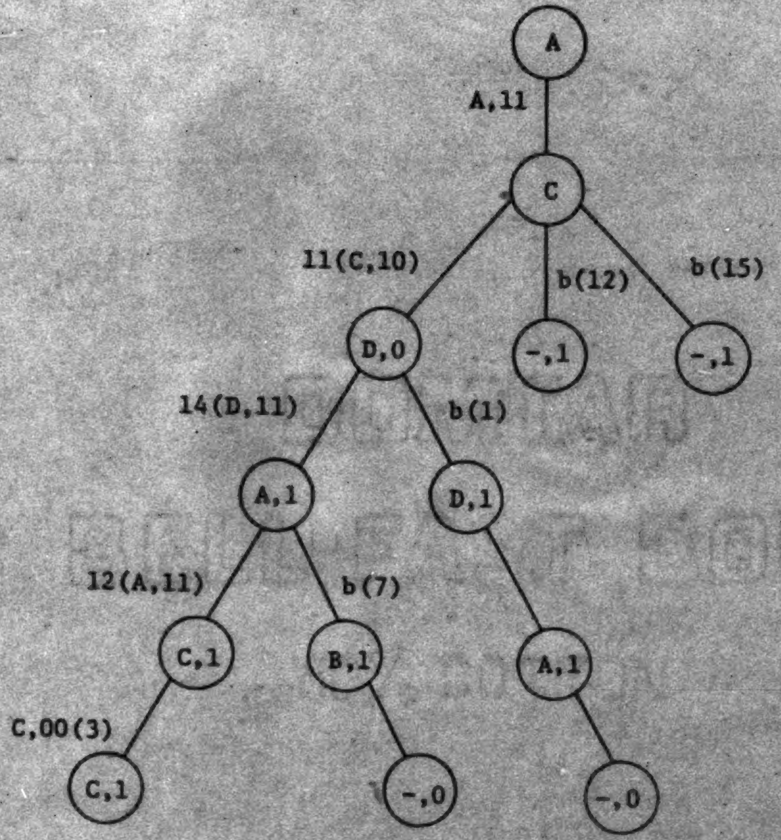

A
A,11
C
11(C,10)
b(12)
b(15)
D,0
-,1
-,1
14(D,11)
b(1)
A,1
D,1
12(A,11)
b(7)
C,1
B,1
A,1
C,00(3)
C,1
-,0
-,0

Fig. 5.2 Testing tree for Example 5.1b

As illustrated by the previous example, Procedures 3.3 and 3.4 are easily adapted for the generation of fault diagnosing sequences. Fault diagnosing sequences with various degrees of diagnostic resolution can be obtained.

Chapter 6

CONCLUSIONS

This thesis is concerned with the problem of testing of sequential circuits. In the first part of this thesis the two basic approaches to fault detection in sequential circuits are described. The first approach is represented by circuit testing methods. The simulation method [24,3] can be used to generate testing sequences of minimal length but requires a large amount of computation and storage. Modifications that significantly reduce the amount of computation required and which yield nearly minimal sequence were proposed. Both the original and modified algorithms were programmed and applied to machine M2.2 considering all stuck type faults. While the original algorithm failed to yield any results after 5 minutes of computation on an IBM 360/67 the modified method yielded the complete testing experiment after only 23 seconds for the same circuit. The length of the sequence was 7 inputs.

The second approach is represented by machine identification methods [13,16,3,9]. This approach requires less computation especially for machines which possess a distinguishing sequence. The sequences generated by this method, however, are much longer than in the previous case. The testing sequence for the same circuit as used above would require at least 60 inputs.

Finally a hybrid method which combines both previous approaches reduces the length of the experiment by eliminating tests for transitions which cannot be influenced by any fault in a given set of faults. Assuming that the set of faults consists of all possible stuck type faults, the length of the sequence for machine M2.2 is 20 inputs.

In the second part of this thesis a general method for the development of testing sequences is proposed. In this method the length of the experiment and amount of computation are reduced by classifying faults into output and state faults. The length of the sequence is further reduced by finding a minimal set of tests which has to be applied to the machine to test for a given set of faults and by utilizing the distinguishing properties of inputs in the a minimal set of tests. The length of the sequence for machine M2.2 is 10 inputs. The storage requirements are also reduced. This occurs because it is only necessary to remember the state table of a good machine along with a test table instead of the state tables of both the good and all faulty machines. This method is shown to be easily adaptable for fault diagnosis with various degrees of diagnostic resolution.

Using the new method, the selection of inputs for a testing sequence is in many cases arbitrary. It is possible that an algorithm for systematic selection of inputs can be found. Such an algorithm could in turn result in further reduction of a testing sequence. To further improve the usefulness of the new testing method, a better procedure for the generation of complete sets of tests for given sets of faults should be found. Ideally such a procedure would be quite general and independent of the circuit architecture and of types of possible faults. Then, once a circuit description along with the possible faults would be known, the test generation could be done on a computer. A satisfactory scheme of describing electrical circuits in a form suitable for computer processing is another problem. More research in this area could prove rewarding.

REFERENCES

[1] Armstrong, D. B., "On Finding a Nearly Minimal Set of Fault Detection Test for Combinational Logic Nets", *IEEE Trans. on Elec. Computers*, vol. EC-15, pp. 66-73, February 1966.

[2] Chang, H. Y., "An Algorithm for Selecting an Optimum Set of Diagnostic Tests", *IEEE Trans. on Elec. Computers*, vol. EC-14, pp. 706-710, October 1965.

[3] Chang, H. Y., Manning, E. G. and Metze, G., *Fault Diagnosis in Digital Systems*, Wiley-Interscience, 1970.

[4] Chiang, A.C.L., Reed, J. S. and Barnes, A. V., "Path Sensitization, Partial Boolean Difference and Automated Fault Diagnosis", *IEEE Trans. on Computers*, vol. C-21, pp. 185-195, February 1972.

[5] Du, M-W. and Weiss, C., D., "Multiple Fault Detection in Combinational Circuits", *Proceedings*, International Symposium on Fault-Tolerant Computing, pp. 120-125, Newton, Mass., June 1972.

[6] Farmer, D. E., "Algorithms for Designing Fault-Detection Experiments for Sequential Machines", *IEEE Trans. on Comp.*, vol. C-22, pp.159-167, February 1973

[7] Flomenhoft, M. J., Si, S., and Susskind, A. K., "Algebraic Techniques for Finding Tests for Several Fault Types", *Proceedings*, International Symposium on Fault-Tolerant Computing, pp. 85-90, Palo Alto, Calif., June 1973.

[8] Fridrich, M., "Fault Detection in Combinational Networks", *PhD Disertation*, Univ. of Alberta, Edmonton, January 1973. Also as *Technical Report TR73-1*, Univ. of Alberta.

[9] Friedman, A. D. and Mennon, R., *Fault Detection in Digital Circuits*, Prentice-Hall, Inc., N. J., 1971.

[10] Friedman, A. D., "Diagnosis of Short Faults in Combinational Circuits", *Proceedings*, International Symp. on Fault-Tolerant Computing, pp. 95-100, Palo Alto, Calif., June 1973.

[11] Gonenc, G., "A Method for the Design of Fault Detection Experiments", *IEEE Trans. on Computers*, vol. C-19, pp. 551-558, June 1970.

[12] Hayes, J. P., and Friedman, A. D., "Test Point Placement to Simplify Fault Detection", *Proceedings*, International Symp. on Fault-Tolerant Computing, pp. 73-78, Palo Alto, Calif., June 1973.

[13] Hennie, F. C., "Fault Detecting Experiments for Sequential Circuits", *Proc. of the 5th Ann. Symp. on Switching Theory and Log. Design*, pp. 95-110, October 1964.

[14] Hennie, F. C., *Finite-State Models for Logical Machines*, John Wiley & Sons,Inc., 1968

[15] Hornbuckle, G. D., and Spann, R. N. "Diagnosis of Single-Gate Failures in Combinational Circuits", *IEEE Trans. on Computers*, vol. C-18, pp. 216-220, March 1969.

[16] Hsieh, E. P., "Checking Experiments for Sequential Machines", *IEEE Trans. on Computers*, vol. C-20, pp. 1152-1166, October 1971.

[17] Kautz, W. H., "Fault Testing and Diagnosis in Combinational Digital Circuits", *IEEE Trans. on Comp.*, pp. 352-366, April 1968.

[18] Kohavi, I. and Kohavi Z., "Variable-Length Distinguishing Sequences and their Application to the Design of Fault-Detection Experiments", *IEEE Trans. on Computers*, vol. C-17, pp. 792-795, August 1968.

[19] Kohavi, I., "Fault Diagnosis of Logical Circuits", *IEEE Conf. Rec. of 1969 Tenth Ann. Symp. on Switching and Aut. Theory*, pp. 166-173, October 1969.

[20] Kohavi, I. and Kohavi, Z., "Detection of Multiple Faults in Combinational Logic Networks", *IEEE Trans. on Computers*, vol. C-21, pp.556-567, June 1972.

[21] Koga, Y. and Hirata, F., "Fault Locating Test Generation for Combinational Logic Networks", *Proceedings*, International Symposium on Fault-Tolerant Computing, pp. 131-136, Newton, Mass., June 1972.

[11] Gonenc, G., "A Method for the Design of Fault Detection Experiments", *IEEE Trans. on Computers*, vol. C-19, pp. 551-558, June 1970.

[12] Hayes, J. P., and Friedman, A. D., "Test Point Placement to Simplify Fault Detection", *Proceedings*, International Symp. on Fault-Tolerant Computing, pp. 73-78, Palo Alto, Calif., June 1973.

[13] Hennie, F. C., "Fault Detecting Experiments for Sequential Circuits", *Proc. of the 5th Ann. Symp. on Switching Theory and Log. Design*, pp. 95-110, October 1964.

[14] Hennie, F. C., *Finite-State Models for Logical Machines*, John Wiley & Sons,Inc., 1968

[15] Hornbuckle, G. D., and Spann, R. N. "Diagnosis of Single-Gate Failures in Combinational Circuits", *IEEE Trans. on Computers*, vol. C-18, pp. 216-220, March 1969.

[16] Hsieh, E. P., "Checking Experiments for Sequential Machines", *IEEE Trans. on Computers*, vol. C-20, pp. 1152-1166, October 1971.

[17] Kautz, W. H., "Fault Testing and Diagnosis in Combinational Digital Circuits", *IEEE Trans. on Comp.*, pp. 352-366, April 1968.

[18] Kohavi, I. and Kohavi Z., "Variable-Length Distinguishing Sequences and their Application to the Design of Fault-Detection Experiments", *IEEE Trans. on Computers*, vol. C-17, pp. 792-795, August 1968.

[19] Kohavi, I., "Fault Diagnosis of Logical Circuits", *IEEE Conf. Rec. of 1969 Tenth Ann. Symp. on Switching and Aut. Theory*, pp. 166-173, October 1969.

[20] Kohavi, I. and Kohavi, Z., "Detection of Multiple Faults in Combinational Logic Networks", *IEEE Trans. on Computers*, vol. C-21, pp.556-567, June 1972.

[21] Koga, Y. and Hirata, F., "Fault Locating Test Generation for Combinational Logic Networks", *Proceedings*, International Symposium on Fault-Tolerant Computing, pp. 131-136, Newton, Mass., June 1972.

[22] Mei, K. C. Y., "Bridging and Stuck-At Faults", *Proceedings*, International Symp. on Fault-Tolerant Computing, pp. 91-94, Palo Alto, Calif., June 1973.

[23] Poage, J.F., "Derivation of Optimum Tests to Detect Faults in Combinational Circuits", *Proc. of the Symp. of Math. Theory of Automata*, pp.483-528, April 24-26, 1962.

[24] Poage, J. F. and McCluskey, E. Y., "Derivation of Optimum Tests Sequences for Sequential Machines", *Proc. of the 5th Ann. Symp. on Switching Theory and Logical Design*, Princeton University, pp. 121-132, Nov. 11-13, 1969.

[25] Ramos, S. and Smith, A. R., "Fault Detection in Uniform Modular Realizations of Sequential Machines", *Proceedings*, International Symposium on Fault-Tolerant Computing, pp. 114-119, Newton, Mass., June 1972.

[26] Reese, R. D. and McCluskey, E. J., "A Gate Equivalent Model for Combinational Logic Network Analysis", *Proceedings*, International Symp. on Fault-Tolerant Computing, pp. 79-84, Palo Alto, Calif., June 1973.

[27] Roth, J. P., "Diagnosis of Automata Failures: A Calculus and a Method", *IBM Journal of Research and Development*, vol. 10, pp. 279-291, July 1966.

[28] Seshu, S. and Freeman, D. N., "The Diagnosis of Asynchronous Sequential Switching Systems", *IRE Trans. on Elec. Comp.*, vol EC-11, pp. 459-465, August 1962.

[29] Seshu, S., "On an Improved Diagnosis Program", *IEEE Trans. on Elec. Comp.*, vol. EC-14, pp. 76-79, January 2965.

[30] Vancura, R. P. and Kime, C. R., "On Numerical Bounds in Diagnosable Systems", *Proceedings*, International Symposium on Fault-Tolerant Computing, pp. 148-153, Newton, Mass., June 1972.

Appendix I.

A Program for the Generation of the Testing Sequences.

The following is a description of a PL/1 program which generates testing sequences using modified Poage's method, described in Section 2.1.1. The program reads in the description of the circuit, number of faults,their locations and initial state. It then finds the input sequences which must be applied to the machine in order to detect every given fault. These sequences are merged into a testing sequence which, when applied to the machine will detect all detectable given faults.

The input data consits of the following:

a) Circuit description:

1) Size card: It contains, in free format:
   number of inputs (NIPTS),
   number of gates (NGATES),
   max. number of inpus per gate (NINGA) and
   number of feedback lines (NFEEDB) .

2) Circuit cards: The circuit is described in the form of a matrix. All wires leading from a gate or an input are considered

as a connection and identified by a unique number. The number of rows of the matrix is the same as the number of gates in the circuit and number of columns is NIPTS+NFEEDB+NGATES, and corresponds to the total number of connections. The connections must be numbered in such a way that inputs have the lowest numbers, feedback inputs come next and then all other connections can be numbered in any order. If the wire n is connected to the gate m there will be the codes appearing in the following two columns entered c - into the m-th row and the n-th column of the matrix and to the m-th row and the m-th column of the matrix, respectively. The codes are:

1 - for input
2 - for output
3 - for feedback input
4 - for all other connections

A - AND gate
O - OR gate
N - NOT gate

If an input, output or feedback input wire is connected to several gates, it can be denoted with 1,2 or 3, respectively, only for one gate , and must be denoted with a 4 for all other gates. This means that if the same input wire is connected to two gates it will be coded with a 1 in the row corresponding to the first gate and with a 4 in the row corresponding to the second gate.

3) Feedback card contains list of conections identified by their numbers which are feeding feedback lines (feedback outputs)

b) <u>Fault description</u>:

1) Number of faults.

2) Fault location and fault type cards: an s-0 fault will be indentified by " 0B " and an s-1 fault by " 1B ". Location of the fault will be given by connection number accompanied by the gate number, i.e., s-1 fault on wire 2 connected to gate 5 will be coded as 1B 2 5.

c) <u>Initial state description</u>:

There is only one card containing intial state of the machine. Actual state assignment values must be given i.e if the experiment should start with the state A which is assigned value 000 the card will contain 000B.

<u>Example A.1:</u> The following is an input data for machine M2.2 The circuit and set of faults are identical to Example 2.2 with connections numbered according to the above rules, and with both s-0 and s-1 faults considered.

```
2 11 3 2
1000N0000C00002
00304A000C00000
0040C0NC0C0C000
4100004A000C000
040C0000N000000
```

```
000040004A00000
4000004000A0000
440C00000C0A000
0003C40400C0000
C00C00000440000
000C04C00044000
13 14
8
0B 1 1 1B 1 1
0B 3 2 1B 3 2
0B 9 6 1B 9 6
0B 11 10 1B 11 10
00B INITIAL STATE
```

Results:

The sequences obtained by this program were nearly minimal in all cases which were verifiable ,i.e., small machines with small fault sets.

Resources:

1) Memory for:

Circuit description : NGATES * (NIPTS+NFEEDB+NGATES) bytes

State tables : NFEEDB*2**NFEEDB*2**NIPTS*NFAULTS+1 bits

Sequential table : 2**((NFAULTS+1)*NFEEDB+1)*(2**NIPTS*1) bits

The actual size of the sequential table is usually much smaller. The practical estimate is

2**(2*NFEEDB)*(2**NIPTS+1)*NFEEDB .

This formula is based on the fact that any fault must be detected if all possible inputs are applied to every state of the machine. There are 2**NFEEDB states with transfer sequence length not exeeding 2**NFEEDB-1. This sequence will have to be repeated 2**NIPTS*2**NFEEDB times. Each transition will require NFEEDB bits to be entered into sequential table.

2) Processing time:

By running the program on an IBM 360/67 the following CPU times were required:

a) 4 state machine, 4 faults, full seq. table: 3.8 secs

b) " " , mod. seq. table: 2.29 secs

c) 4 state machine, 8 faults, full seq. table: 9.29 secs

d) " " , mod. seq. table: 4.02 secs

e) 4 state machine, 38 faults, full seq. table: not known, but more than 5 min.

f) " " , mod. seq. table: 23.94 secs

The processing time will depend on number of states of machine tested and on number of faults considered. It will be proportional to 2*2**NFEEDB*2**NIPTS.